Programmers Guide to DDGui

(Second Edition)

A programmers guide to using the DDGui GUI system that comes with GLBasic

By Nicholas Kingsley

DDGui was written by Gernot Frisch

ISBN #: 978-1-4475-8191-8

Designed By Nicholas Kingsley

<u>Revision : 2.0.0.0</u>

For errors and omissions, I can be contacted at njk@un-map.com

Index

Part 1

DDgui

Introduction

What DDGui is

DDGui is designed to allow each platform supported by GLBasic to have a GUI system that looks, acts and used in the same way.

DDGui allows the creation of buttons, sliders, edit boxes, check and radio buttons in a window that can be moved

What DDGui isn't

DDGui isn't a fully fledged GUI system – there are many things that haven't been implemented within DDGui – for example positioning is done relative to other other items, as opposed to positioning to X and Y coordinates

Features of DDGui

DDGui has the following features :

- Windows
- Static text
- Buttons
- Sliders
- Toolbars
- Check boxes
- Radio Buttons
- List Box
- Combo
- File Dialog
- Text Boxes
- Tabs
- Frames
- Spacing

There are also some helper functions :

- Dialog Window
- Colour Picker
- Inserting and Deleting text
- Retrieving text
- Hiding and Showing

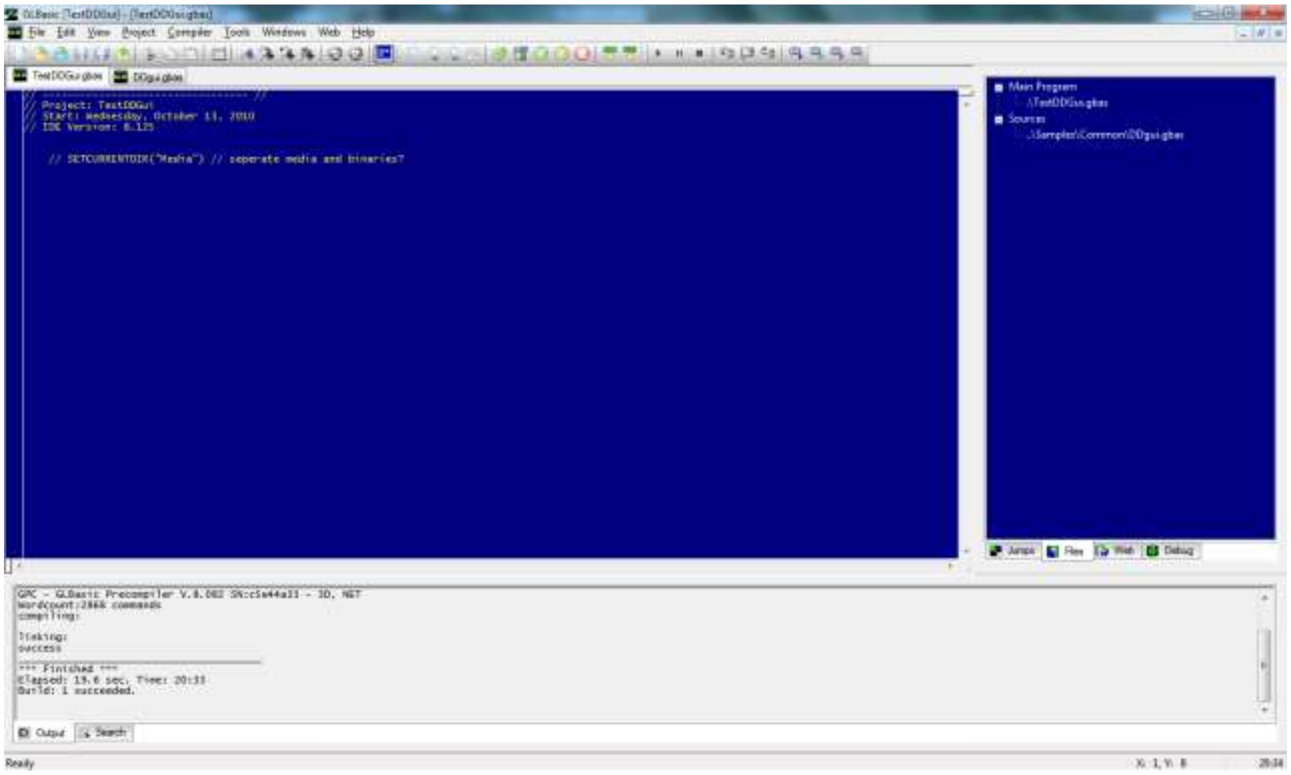
Other areas :

- User-defined routines
- Positioning
- Read-Only
- Tool-tips
- Input
- Progress Bar

<u>Using DDGui</u>

To start using DDGui in your programs, you need to add the DDGui.gbas file to your program. You can do this by adding the file from the Samples → Common to your project.

The file will then be added to the sources list. Your project layout could then look like :


GLBasic [TestDDGui] - [TestDDGui.gbas]
File Edit View Project Compiler Tools Windows Web Help
TestDDGui.gbas
DDgui.gbas
Main Program
.\TestDDGui.gbas
Sources
..\Samples\Common\DDgui.gbas
Jumps Files Web Debug
Output Search
Ready

```
// ------------------------------------ //
// Project: TestDDGui
// Start: Wednesday, October 13, 2010
// IDE Version: 8.125

  // SETCURRENTDIR("Media") // seperate media and binaries?
```

```
GPC - GLBasic Precompiler V.8.082 SN:c5a44a33 - 3D, NET
Wordcount:2868 commands
compiling:

linking:
success
_______________________________
*** Finished ***
Elapsed: 13.6 sec. Time: 20:33
Build: 1 succeeded.
```

You could compile the project like this, but it certainly won't do much, so the first thing we need to do is create a window, using the following :

```
DDgui_pushdialog(0,0,100,100,FALSE)
```

This creates a window 100 by 100 pixels in height, positioned at 0 pixels across and 0 pixels down from the top left corner of the screen. We make sure that the window is not centred to the middle of the screen

To keep showing the window, we need to call the show window command in a loop, which in this case is `DDgui_show`.

This command takes one parameter which determines whether all created windows are displayed, or just the last one. Like all graphics commands in GLBasic, `SHOWSCREEN` has to be called to swap screen buffers and update the display.

So, the routine will be :

```
WHILE TRUE
        DDgui_show(FALSE)
        SHOWSCREEN
WEND
```

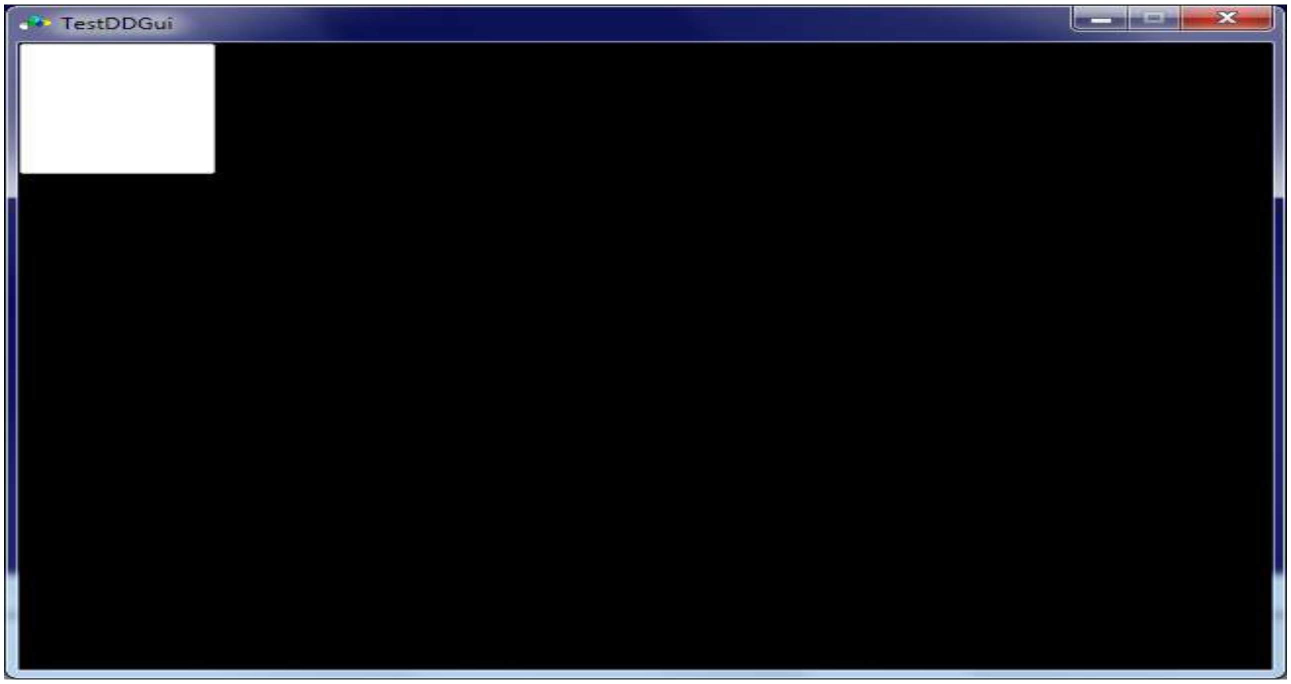
The full program looks like :

```
DDgui_pushdialog(0,0,100,100)
WHILE TRUE
        DDgui_show(FALSE)
        SHOWSCREEN
WEND
```

The full code is :

```
DDgui_pushdialog(0,0,100,100)
WHILE TRUE
        DDgui_show(FALSE)
        SHOWSCREEN
WEND
```

And the results look like :



As mentioned in the *GLBasic Programmers Reference Guide*, the fault font makes text hard to read, so its always a good idea to create a readable font at this point, and place it in the with the executable (non-Mac), or for the Mac, place in the Media directory and manually load the font.

Text Display

From V9.054 all text display by default uses proportional text printing. This can be disabled (or re-enabled) by using `DDgui_UpdateFont`.

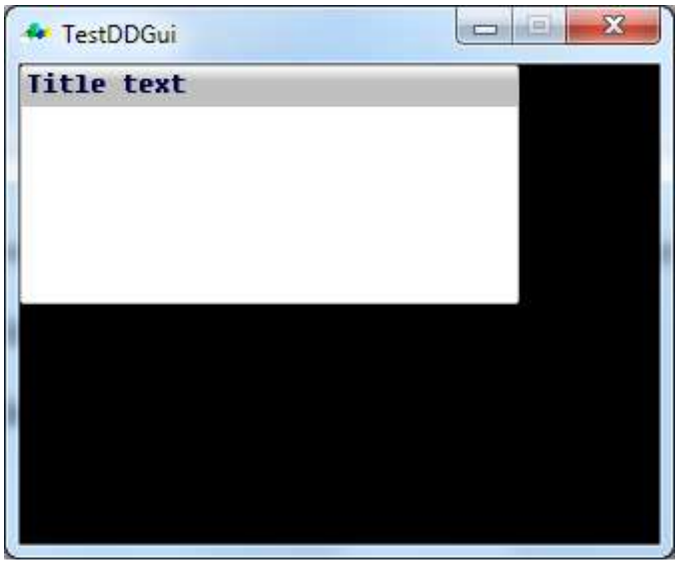
Windows Abilities

Titles

Windows can have titles, which can be set using DDgui_set, with an empty ID value :

```
DDgui_set("","TEXT","Title text")
```

Produces the following title :

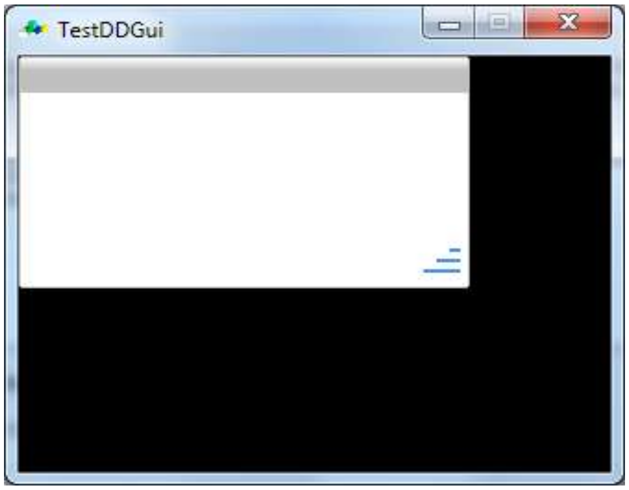


Allowing re-sizing

By default, a DDgui window can't be re-sized by the user. This ability can be activated by calling

```
DDgui_set("","SCALEABLE",TRUE)
```

This produces a window with a “gripper” (blue chevrons) on the bottom right of the window, which allow the window to be changed horizontally and/or vertically :



Allowing movement

You can allow the window to be moved by using :

```
DDgui_set("","MOVEABLE",TRUE)
```

Feedback

You can get feedback about the re-sizing and movement status by using `DDgui_get`, which is discussed in the reference section.

Multiple Windows

Multiple windows can be created with widgets. However, only the last created one will receive user input. In addition, there is currently no way of selecting another window.

ID's

All widgets (except for windows) require a unique string ID – this enables widgets to be uniquely identified and manipulated.

Static Text

To display static text, you would use :

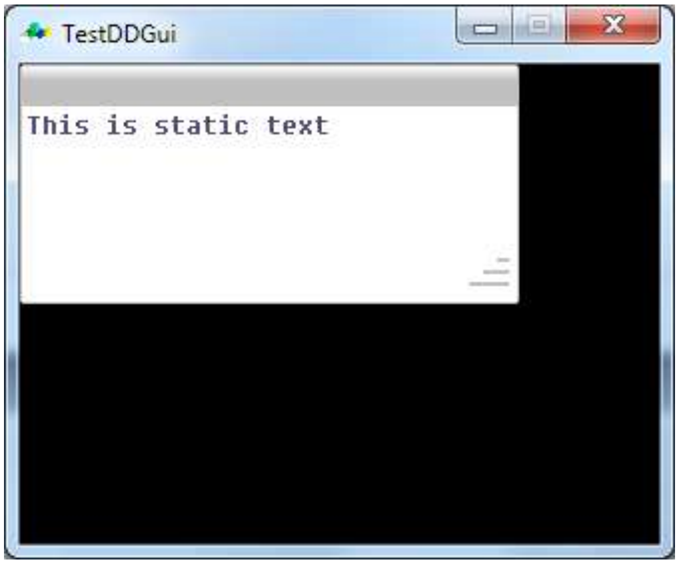
DDgui_widget(id$, text$ *[, width%, height%]*)

with **id$** being a unique identifier, and **text$** is the text to be displayed.

width% and **height%** are optional. If these are given, the the widget will be created with the horizontal size defined by **width%** and the vertical size by **height%**.

If these values aren't given, the the widget with take the horizontal size of the text length and the font height.

Static text looks like :



This is static text

Buttons

Buttons allow the user to select an option, and are created with :

`DDgui_button(id$, text$, [, width%, height%])`

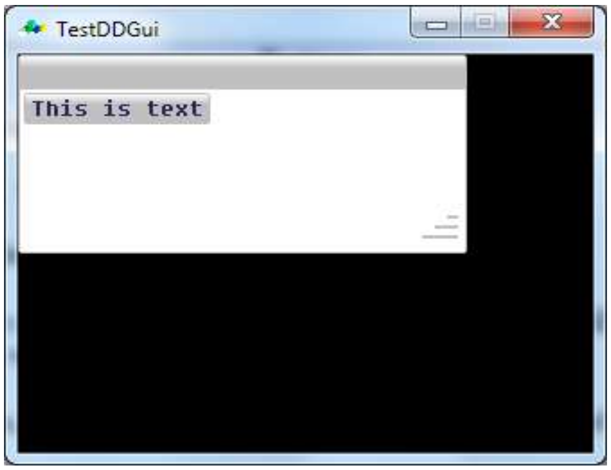
with **id$** being a unique identifier, and **text$** is the text to be displayed.

width% and **height%** are optional. If these are given, the the widget will be created with the horizontal size defined by **width%** and the vertical size by **height%**.

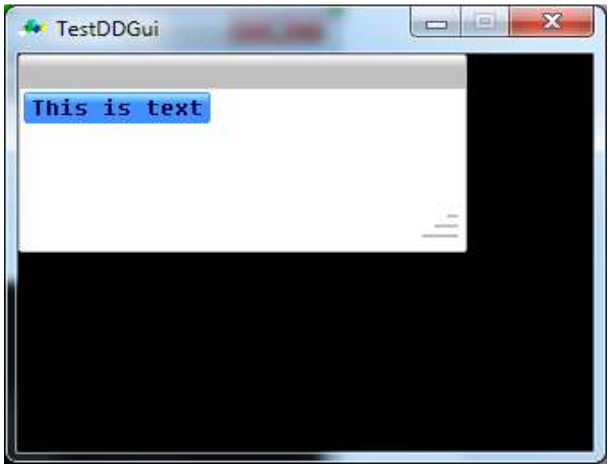
If these values aren't given, the the widget with take the horizontal size of the text length and the font height.

A button has two graphic modes. One when the mouse pointer is over it, and one when it isn't :

Mouse NOT over button



Mouse OVER button



Buttons can also have sprites in them. This is done by setting **text$** to "SPR_B" and appending the sprite index to it

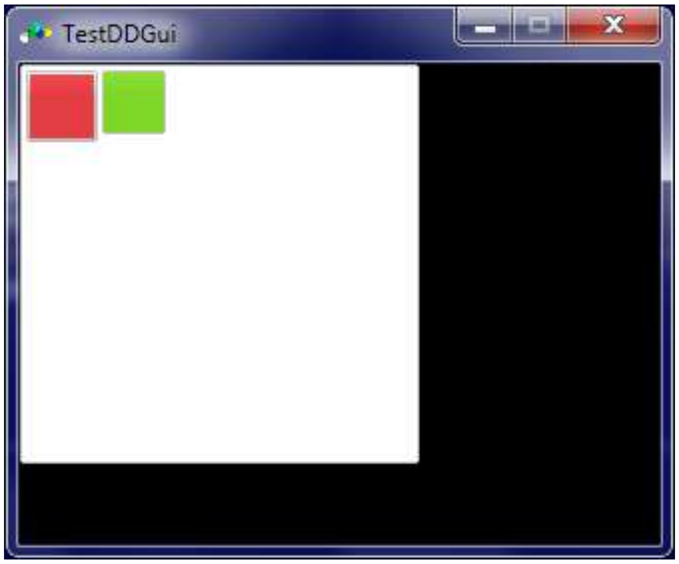
With SPR_B, the sprite size (if **width%** and **height%** are 0) will be used to calculate the size of the button.

In addition, a button can have a filled colour. This is done by setting **text$** to "SPR_C" and appending an integer colour to it. The size of the button created is based on the **width%** and **height%** values – if either of these are 0, then the default width (or height) is 32

SPR_C buttons, when pressed will open the Colour Dialog window.

This graphic shows a red graphic being displayed (using SPR_B), whilst the green button was created using SPR_C :

```
DDgui_button("test1","SPR_B0",0,0)
DDgui_button("test2","SPR_C121712",0,0)
```



Sliders

No, this isn't to do with the American Sci-Fi program – sliders are a horizontal bar that the user can can to specify a value.

Sliders can be created with :

```
DDgui_slider(id$, value [,width%, height%])
```

with **id$** being a unique identifier, and **value** is the default value to be used.

width% and **height%** are optional. If these are given, the the widget will be created with the horizontal size defined by **width%** and the vertical size by **height%**.

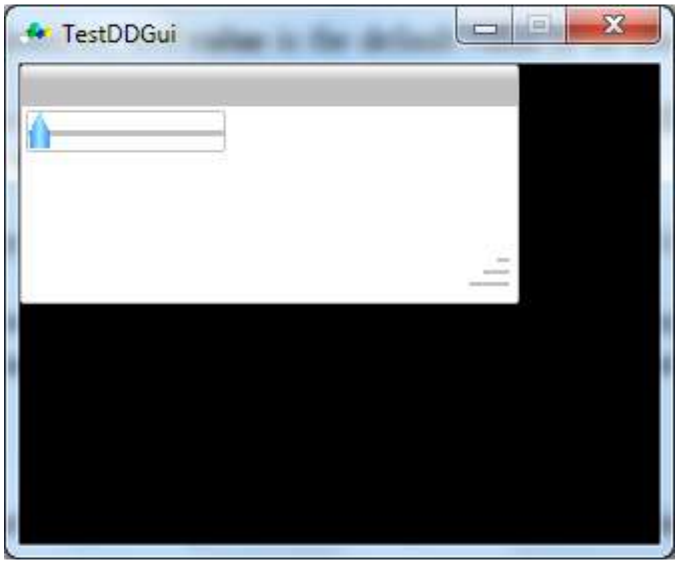
By default, the slider range is between 0.0 and 1.0, with a step value of 0.01

The minimum range can be changed using DDgui_set and a MINVAL parameter, whilst the maximum value can be changed using DDgui_set and a MAXVAL parameter

Note :

- You need to make sure that the pointer position is always between the minimum and maximum range

A slider looks like :



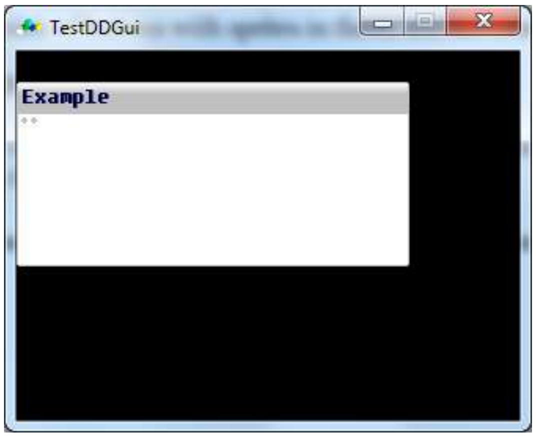
Toolbars

Toolbars allow quick creation of buttons with sprites in them, and are created using :

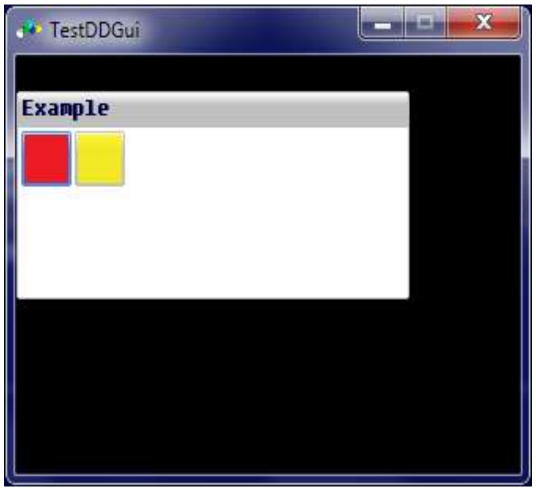
```
DDgui_toolbar(id_buttons$[], id_sprites[])
```

The **id$_buttons$[]** array contains the list of ID's that will be used to detect button presses, while **id_sprites[]** array contain the sprite indexes of the sprites to be used.

These sprites should be loaded before hand – if not, all you will see are small dots :



With sprites loaded, the list would look like :



Note that these buttons are slightly different than sprites in normal buttons. The button with the mouse over it always has a blue border around it.

Which button is pressed can be detected using DDgui_get.

Check Boxes

Check boxes allow the user to activate or de-activate an option, and is created using :

`DDgui_checkbox(id$, text$, [, width%, height%])`

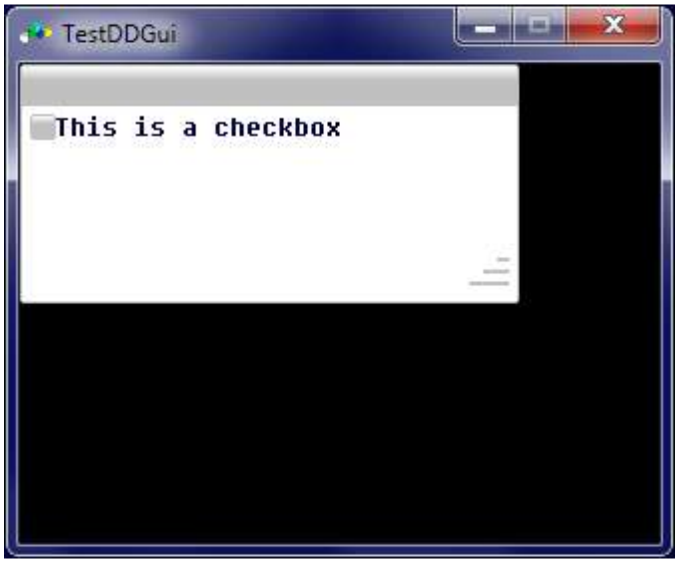
with **id$** being a unique identifier, and **text$** is the text to be displayed.

width% and **height%** are optional. If these are given, the the widget will be created with the horizontal size defined by **width%** and the vertical size by **height%**.

If these values aren't given, the the widget with take the horizontal size of the text length and the font height.

By default a check box is un-ticked. A check box can be pro grammatically activated or de-activated by using `DDgui_set`. The check box status can be read by using `DDgui_get`

A check box looks like :



Radio Buttons

A radio button allows the user to select one of a group of options. A radio button should be used when 2 or more options are needed.

A radio button is created by :

`DDgui_radio(id$, text$ [, width%])`

with **id$** being a unique identifier, and **text$** is the text to be displayed. The text contains all the options to be displayed, with each separated by "|"

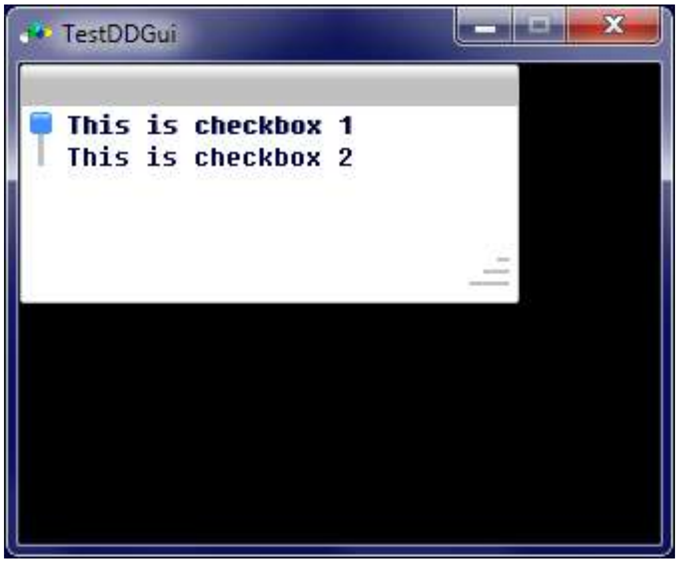
width% is optional. If this is given, the the widget will be created with the horizontal size defined by **width%**.

If this values isn't given, the the widget with take the horizontal size of the largest text length

By default the first line will be checked. This can be changed using `DDgui_set`

Which radio button is selected can be detected using `DDgui_get`

A group of radio buttons look like :



List Box

A list box allows the user to select text from a static, multiple line text box.

A list box is created by :

```
DDgui_list(id$, text$ [,width%, height%])
```

with **id$** being a unique identifier, and **text$** is the text to be displayed. The text contains all the options to be displayed, with each separated by "|"

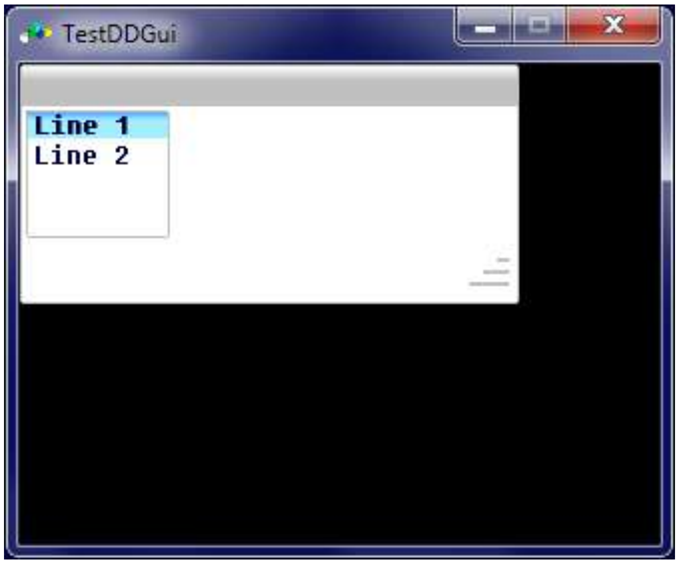
width% and **height%** are optional. If these are given, the the widget will be created with the horizontal size defined by **width%** and the vertical size by **height%**.

If these values aren't given, the the widget with take the horizontal size of the text length and the font height.

By default the first line will be checked. This can be changed using `DDgui_set`

Which line is selected can be detected using `DDgui_get`

A list box looks like :



The vertical size of each line is controlled by the `gDDguiMinControlDimension%` global variable.

Combo Box

A combo box is DDgui's equivalent of a drop-down box, and is called with :

`DDgui_combo(id$, text$ *[, width%, height%]*)`

id$** is the ID for the widget, **text$ is the list of items that could be selected. Each item should be separated by a "|"

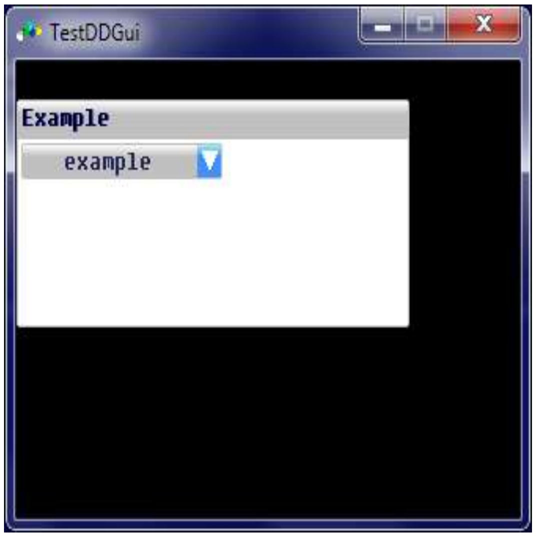
width%, if present, sets the width of the drop-down button. If this value is not present, then the width will be set to the length of the largest item in the list.

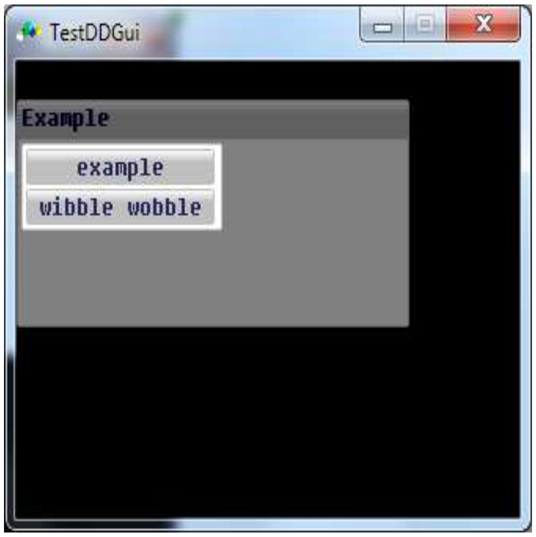
height%, if present, sets the height of the drop-down button. If this value is not present, then the height will be height of the current font being used.

For example using :

```
DDgui_combo("test","example|wibble wobble",0,0)
```

Produces :





The default item displayed is the first item in the list

File Dialog

A file dialog is a window that allows you to enter (or select) a file for loading or save.

A file dialog is used by :

```
result$ = DDgui_FileDialog$(open% [, filter$])
```

open% denotes whether you have opened the dialog window for read (open% = **TRUE**) or writing (**open%** = **FALSE**)

filter$ is an extension which filters any file list to those that have the given extension, if given.

If **filter$** is not present, then all files will be listed.

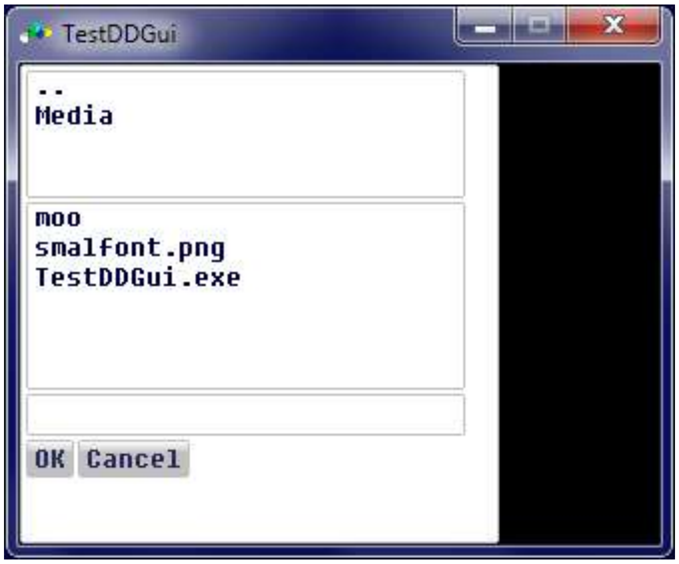
If **open%** = **FALSE**, then a given filename will have the extension stored in **filter$** appended to it (if present). If **filter$** is not present, then no extension will be added

This function returns the filename selected or entered.

Notes :

- If the file dialog is set for writing, then an empty file with the given filename will be created in the current working directory.

A file dialog window looks like :



File selection can also be done using DDgui_file :

It's used by :

`DDgui_file(id$, text$, filter$` *`[, width%, height%]`*`)`

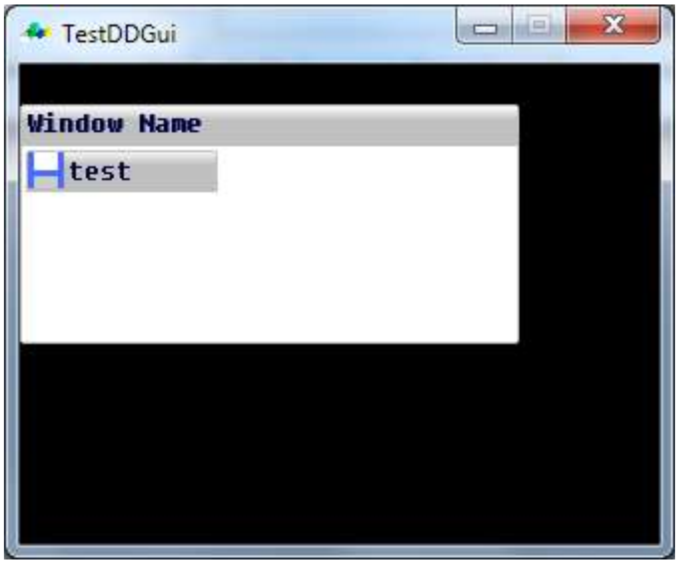
id$** is the widget ID, **text$ is some text that will be displayed to the right of the button.

filter$ is a file filter which will filter the file directory listing to the extension that is held in this parameter.

width% and **height%**, if given, will set this widget to the given width and height. If **width%** is not present, then the font width (based on the length of text$) will be used.

height%, if not present, will use the font height to calculate the height of the widget

The widget looks like :



Text Boxes

An text box allows the user to enter text. Escape characters are replaced by a yellow block, but in the case of a TAB key press it is replaced by a space.

A text box is define by :

`DDgui_text(id$, text$ [,width%, height%])`

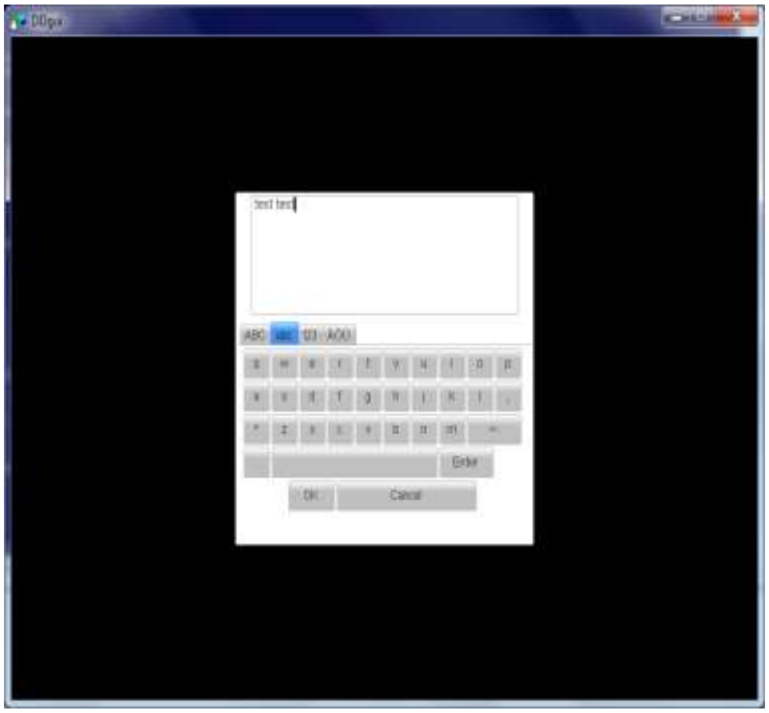
with **id$** being a unique identifier, and **text$** is the text to be displayed. Text boxes are usually for single line entry although multiple lines can be used by either the user pressing RETURN, or by putting in the "\n" escape character before the new line,

width% and **height%** are optional. If these are given, the the widget will be created with the horizontal size defined by **width%** and the vertical size by **height%**.

If these values aren't given, the the widget with take the horizontal size of the text length and the font height.

Additional (or new text) can be added using `DDgui_set$` and retrieved by using `DDgui_get$`

A text box (with onscreen keyboard) looks like :



In GLBasic V9.052 onwards, there is a new text box which is designed to only take one line. With these text boxes, the TAB key moves the caret to the next single line entry box.

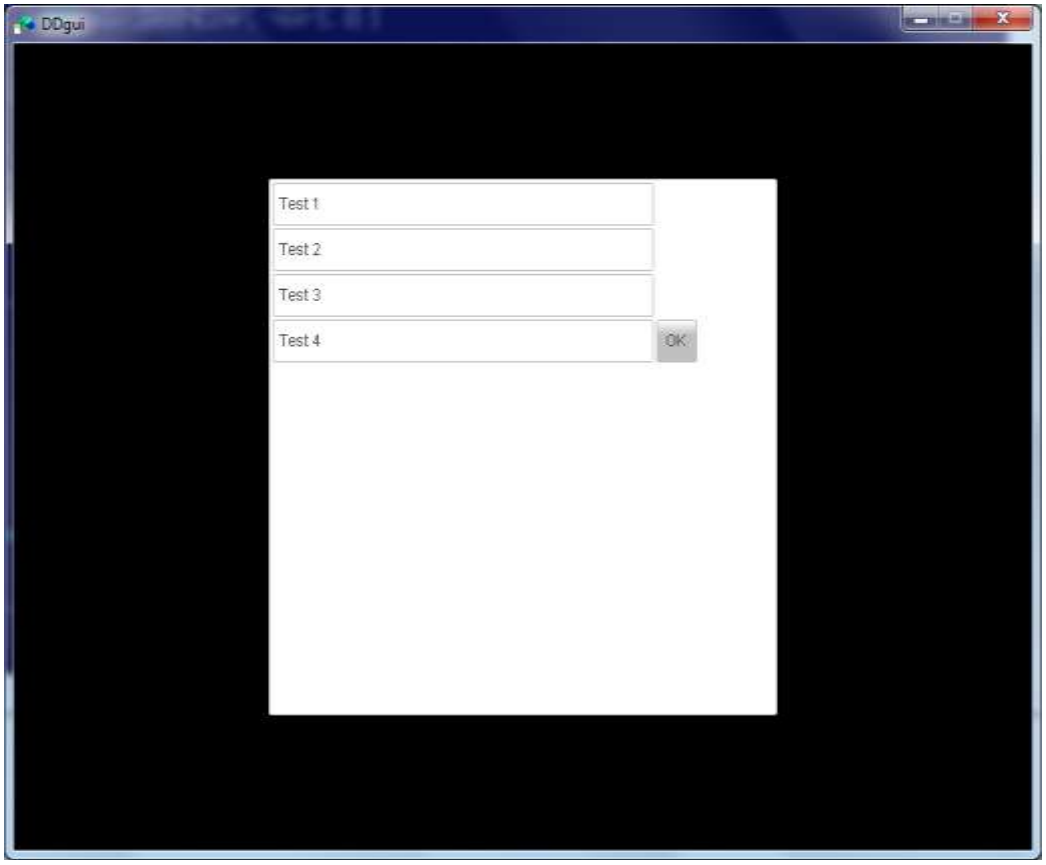
These text boxes are defined by :

`DDgui_singletext(id$, text$ [,`*`width%=0`*`])`

with **id$** being a unique identifier, and **text$** is the text to be displayed. Pressing RETURN moves to the next `DDgui_singletext` text box

width% and **height%** are optional. If these are given, the the widget will be created with the horizontal size defined by **width%** and the vertical size by **height%**.

Single text boxes look like :



Tabs

Tabs allow DDGui widgets to be grouped together in their own window area.

A tab is created by :

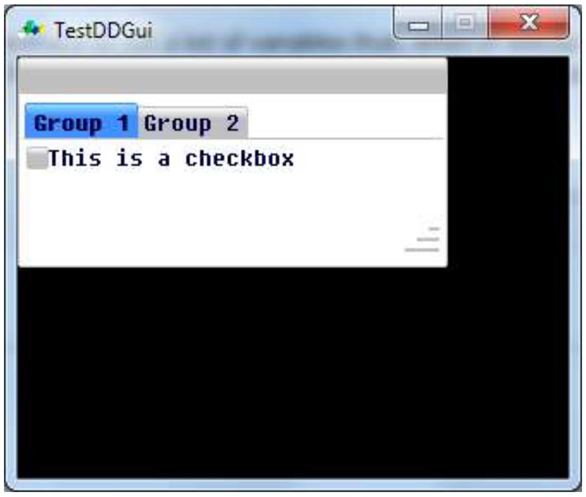
```
DDgui_tab(id$, details$)
```

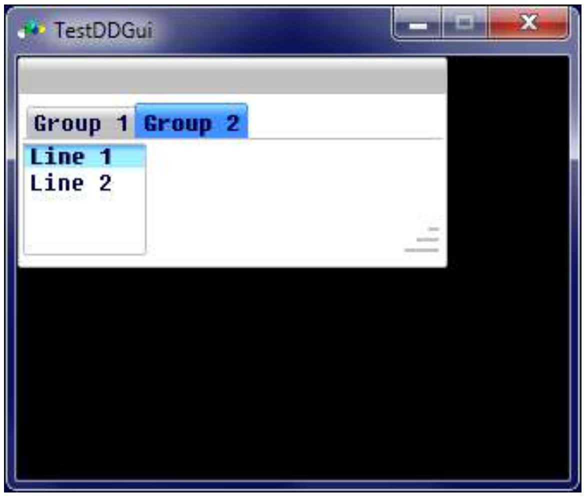
id$** is the id text for the tab, and **details$ contains a list of tab names and widgets that are in the tab. Multiple tabs are separated by a “|” character, so the format is :

tab name,widgets list|tab name,widgets list etc.

Which tab is selected can be read from DDgui_get.

Tabs look like :





<u>Frames</u>

Frames are used to group or detail items in a more aesthetically pleasing way.

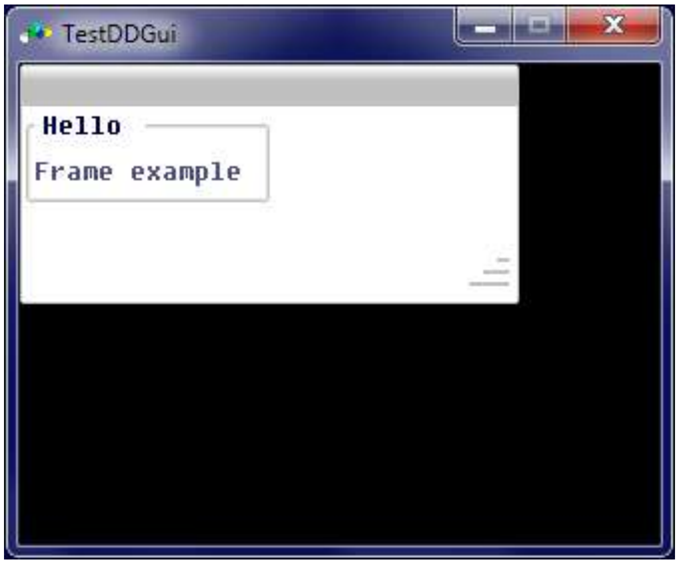
They are created using :

```
DDgui_framestart([id$, caption$, width%])
DDgui_frameend()
```

id$** is the id name you want to give the frame, **caption$ is the title you want to give the frame. If no title is given then an unbroken rectangle is drawn.

width% denotes the width of the frame. If this value isn't given then the width of the largest widget is used to calculate the width of the frame

A frame looks like :



Spacing

As well as setting widgets to a fixed size, you can also introduce spaces by using the DDgui_space function :

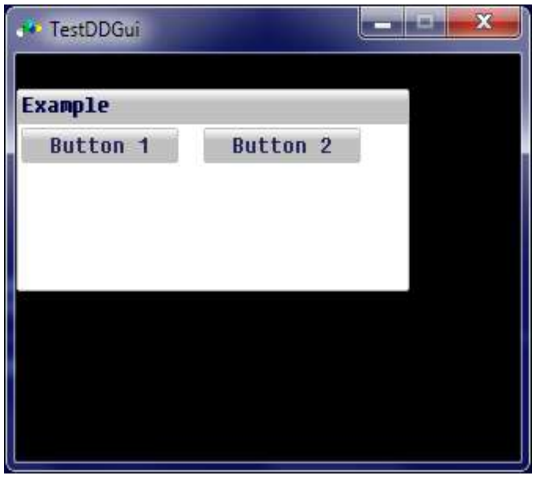
`DDgui_spacer(`*`[width%, height%]`*`)`

If **width%** is given, then the next widget will attempt to be positioned after the given number of horizontal pixels. A large **width%** value will introduce vertical spacing

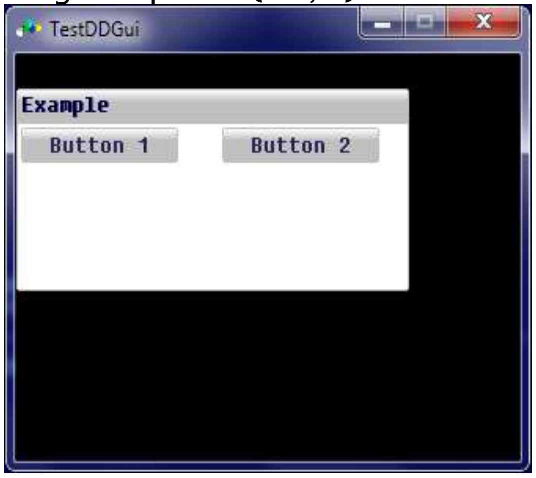
A **height%** value usually has no affect on spacing.

Some examples :

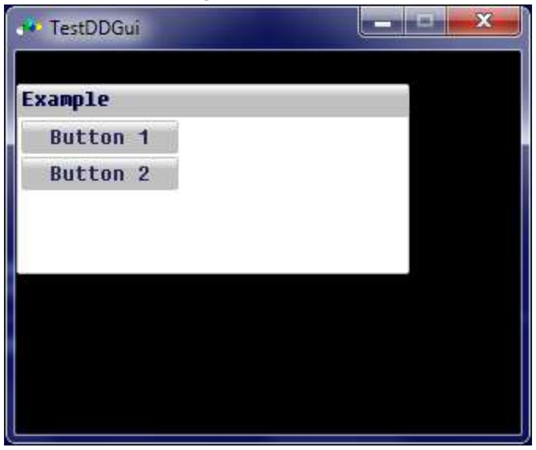
`DDgui_spacer(12,0)`



`DDgui_spacer(24,0)`



`DDgui_spacer()`



`DDgui_spacer` cannot be added to a tab, as it has no ID.

Dialog Window

A general purpose dialog window (which allows the user to select either one or one of two buttons) can be created to display a message or request that the user respond to a query.

A dialog window can be created with :

```
result% = DDgui_msg(text$,yesNo% [,title$])
```

text$ is the text to be displayed. The size of the window is limited, so this text should be kept to a minimum

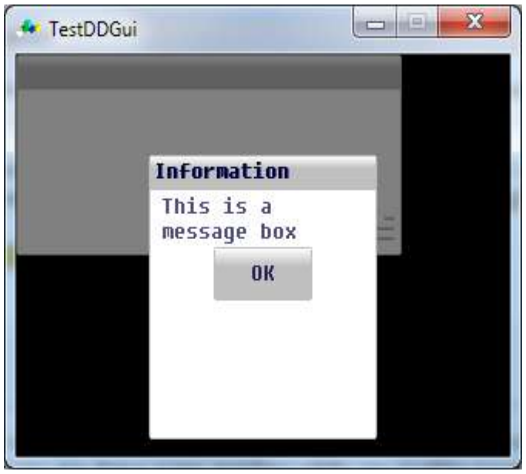
yesNo% indicates whether there should be a simple OK button (**yesNo%** = **FALSE**) or whether there should be two buttons – one with "Yes" and one with "No" (**yesNo%** = **TRUE**)

title$, if present, is the title of the window. If its not present, then the default of "Information" is used.

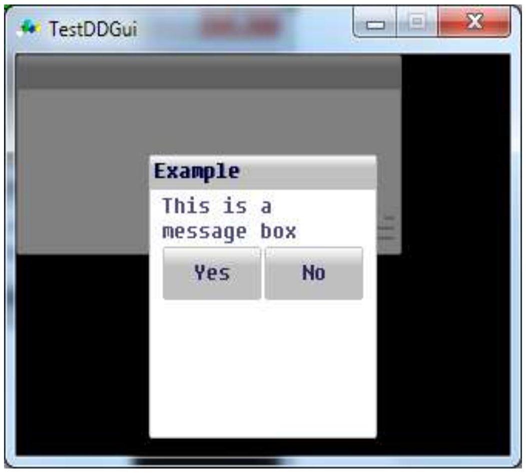
This function returns **TRUE** if the YES or OK button is selected, and **FALSE** if the NO button is selected.

Window with

yesNo% = **FALSE**



yesNo% = **TRUE**



Colour Picker

The colour picker allows the user to choose a colour by modifying the individual components (red, green and blue). HSV and HSL selection would need to be implemented by the user.

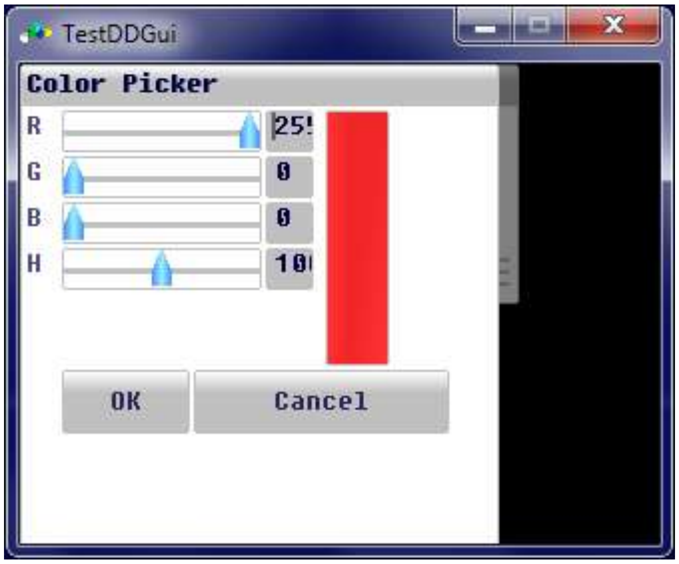
The colour picker is called with :

```
colour% = DDgui_ColorDlg(colour%)
```

`colour%` is the initial colour that will initialise the colour picker. This function returns the colour selected by the user.

If the user has cancelled the window, then the return colour will be the same as that which initialised the function.

The colour picker looks like :



Inserting and Deleting Text

List Boxes

With list boxes, there are two functions used to add and remove lines of text. These are :

```
DDgui_insertitem%: id$, text$, before_index%
```

id$** is the ID of the list box, **text$ is the text to be added, and **before_index%** is the position that you want to insert the text.

If **before_index%**>=0 then the text will be inserted at that line position – don't forget that line numbering starts at 0!

If **before_index%**<0 then the text will be appended to the end of the list

Removing lines from list boxes is done with :

```
DDgui_deleteitem%: id$, index%
```

`id$` is the ID of the list box, and `index%` is the line that you want to delete. If this value is < 0 then everything will be deleted.

Edit Boxes

With edit boxes, you would need to use `DDgui_get` to get the text from the edit box, split into lines, delete or append as appropriate and then put the new text back in using `DDgui_set`

Retrieving Text

Text is retrieved using DDgui_get$ with the id of the widget you want to retrieve the data from, and the parameter of "TEXT", for example :

```
DEBUG DDgui_get$("result","TEXT")
```

With most widgets, the data returned is that same as that entered. However, with list boxes, the data returned is slightly different, in that each line is separated with a "|".

For example :

```
DDgui_list("test","a|B|C",100,100)
DEBUG DDgui_get$("test","TEXT")
```

Produces :

a|B|C

Hiding and Showing

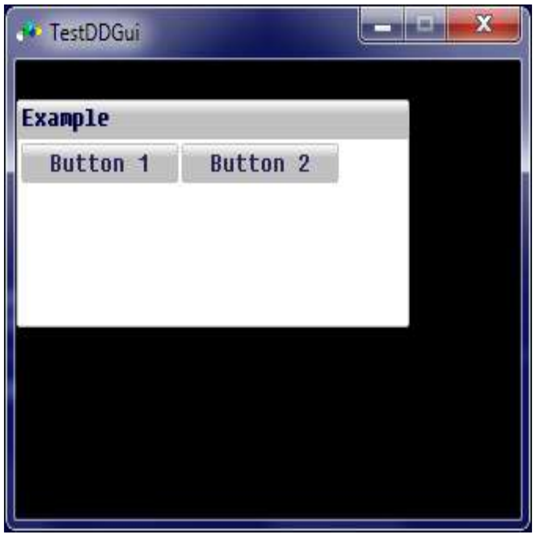
Widgets can be hidden and shown using :

`DDgui_hide(id$ [, hide%])`

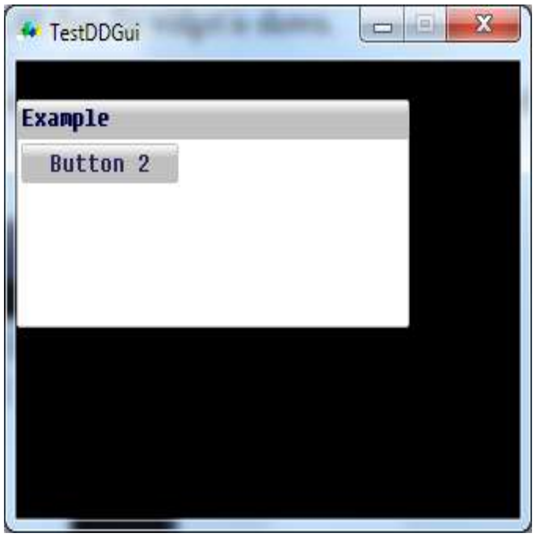
id$ is the ID of the widget, and hide% shows or hides a widget. If **hide%** = **TRUE** then the widget is hidden. If **hide%** = **FALSE** then the widget is shown.

If a widget is hidden then all other widgets next to it will be re-positioned :

With both buttons shown



With Button 1 hidden



Focusing

New in GLBasic 9.054 is the ability to move backwards or forwards through either a text or single text widget. This is done by using :

`DDgui_advancefocus(`*`[direction% = -1]`*`)`

This moves the cursor to either the next text or single text widget (**direction% > 0**) or the previous widget (**direction% < 0**)

`Ddgui_setfocus(id$)`

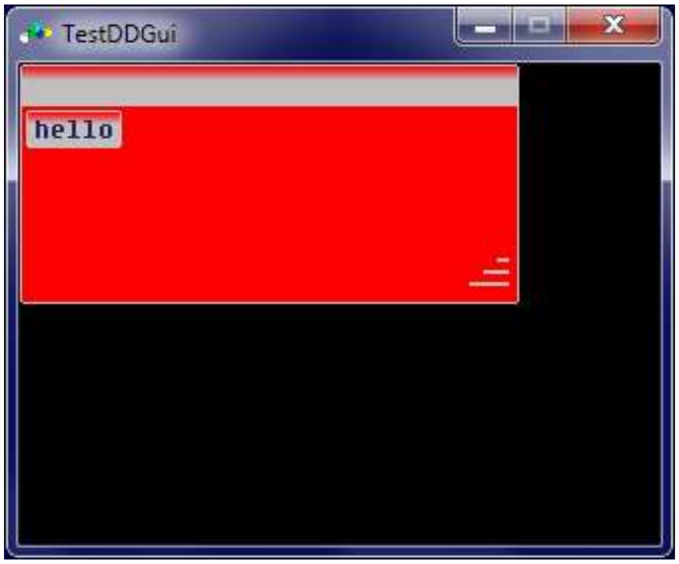
This moves the cursor to the widget defined by **id$**. Again, this only works with text or single text widgets.

User-Defined Routines

DDGui allows a limited amount of customisation, enabling the look of a window to be changed. These are set by using no id parameter in DDgui_set command :

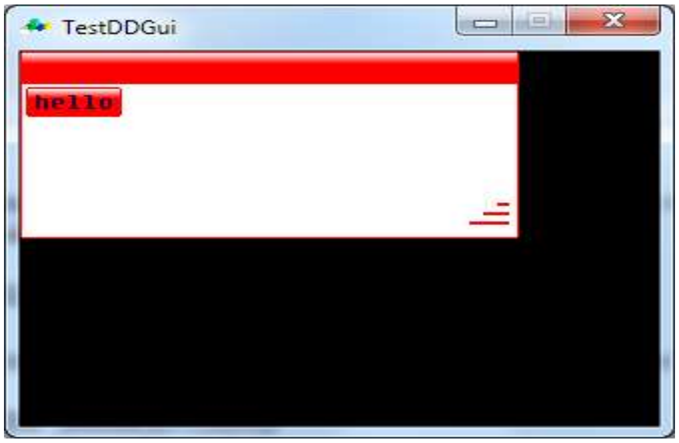
```
DDgui_set("","COL_BRIGHT",RGB(255,0,0))
```

This sets the background colour of the window, and the top part of things like buttons



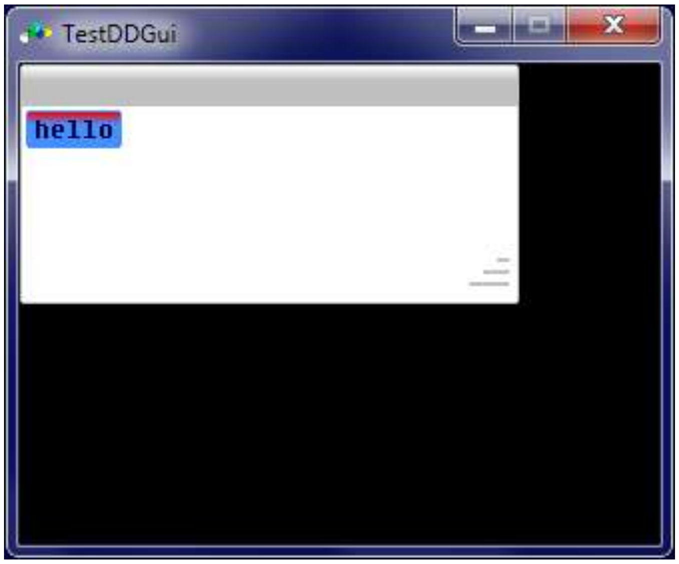
```
DDgui_set("","COL_NORM",RGB(255,0,0))
```

This sets the title colour of the window and the main part of, say, a button when there is no mouse pointer over it



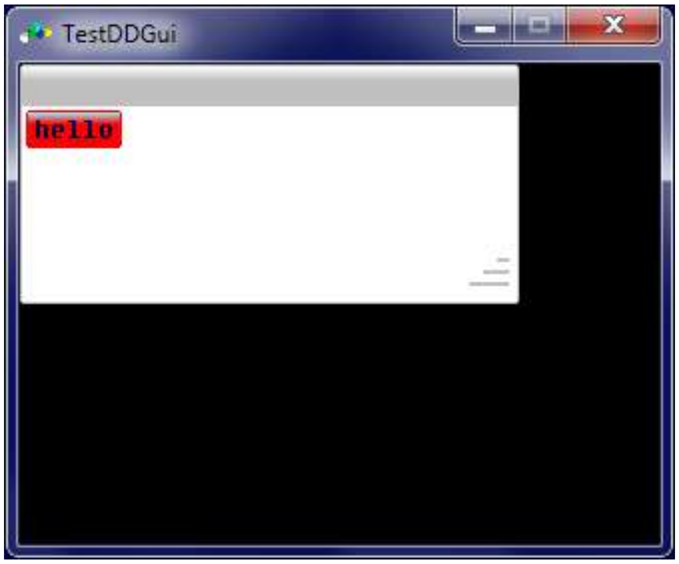
```
DDgui_set("","COL_HOVER_BRIGHT",RGB(255,0,0))
```

This sets the top part of, say, a button graphic when the mouse is over it



```
DDgui_set("","COL_HOVER_NORM",RGB(255,0,0))
```

This sets the colour of the main body of, say, a button, when a mouse pointer is over it



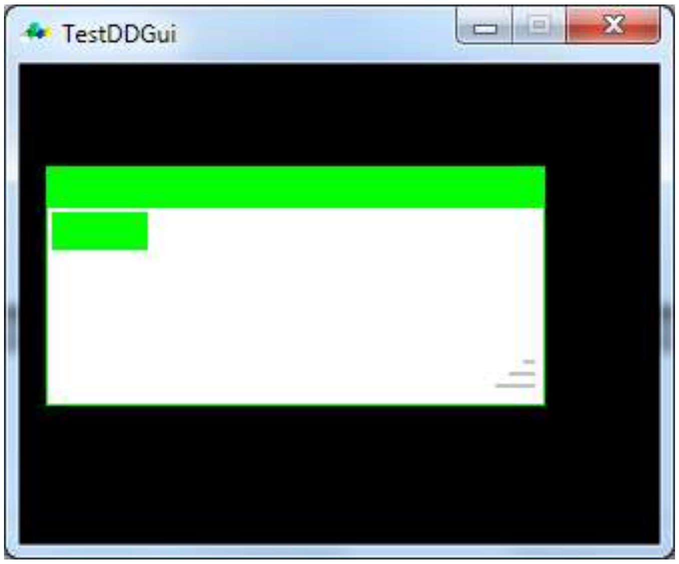
```
FUNCTION DDgui_backrect: x%,y%,dx%,dy%, col%
```

This function allows you to overwrite the colour of the border of the window, the title bar and the main area of widgets like buttons.

For example, using the following :

```
FUNCTION DDgui_backrect: x%,y%,dx%,dy%, col%
        DRAWRECT x%,y%,dx%,dy%,RGB(0,255,0)
ENDFUNCTION
```

produces :



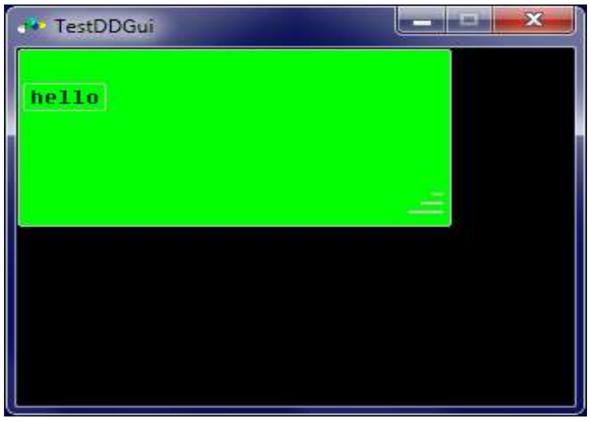
FUNCTION DDgui_backgnd%: col1%, col2%, x%, y%, dx%, dy%

This allows you to override the background colour of windows and widgets

This example :

```
FUNCTION DDgui_backgnd%: col1%, col2%, x%, y%, dx%, dy%
        DRAWRECT x%,y%,dx%,dy%,RGB(0,255,0)
ENDFUNCTION
```

Would produce a green background :



```
DDgui_userdefined(id$, user_type$, width%, height%)
DDgui_draw_user%: BYREF id$, width%, height%, ytop%
DDgui_handle_user%:BYREF id$,mx%,my%,b1%,b2%
```

These three need to be used together, and allow user-defined widgets to be created and processed.

DDgui_userdefined creates a user-defined widget. **id$** gives the widget an ID, **user_type$** defines a type for the widget, and **width%** and **height%** is the width and height of the widget.

DDgui_draw_user draws your user-defined widget. As the routine will let your widget have focus, you perform all drawing from 0,0 coordinates.

id$ is the ID of the widget, **width%** and **height%** are the width and height of the widget and **ytop%** is the vertical position of the widget (which is usually 0)

DDgui_handle_user handles all mouse input for your widget. **id$** is the ID of the widget, **mx%** and **my%** are the mouse X and Y coordinates, relative to the top-left corner of the widget.

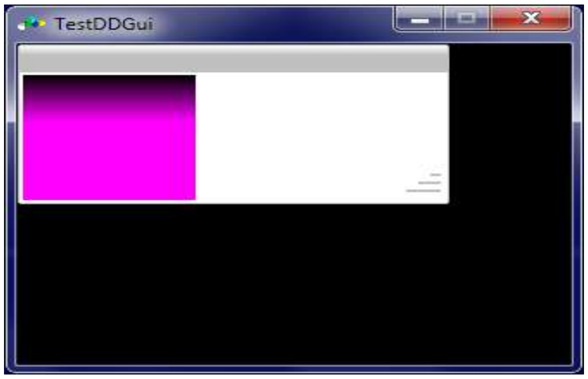
It should be noted that these coordinates can be negative or can be updated outside the window area.

b1% and **b2%** will be **TRUE** if the left or right mouse button within the widget area.

Code for user-defined widget could look like :

```
DDgui_userdefined("test","moo",100,100)

FUNCTION DDgui_draw_user%: BYREF id$, width%, height%, ytop%
        DDgui_backgnd(RGB(0,0,0), RGB(255,0,255), 0,0,width, height)
ENDFUNCTION

FUNCTION DDgui_handle_user%:BYREF id$,mx%,my%,b1%,b2%
        DEBUG id$+" "+mx%+" "+my%+" "+b1%+"\n"
        IF b1%
                DEBUG "Button pressed\n"
                KEYWAIT
        ENDIF
ENDFUNCTION
```

which produces :



With the text "Button pressed" displayed in the Output area in the editor (see GLBasic User Guide for more information) when the user presses the left mouse button within the purple area.

In addition, there are several `GLOBAL` variables than can be used to adjust how DDGui works :

`DDGUI_AUTO_INPUT_DLG`

If set to **`TRUE`**, then a keyboard window will be created at the bottom of the screen when the user clicks on a text box. This value is set to **`TRUE`** for smartphones and some other platforms.

`DDGUI_IN_INPUT_DLG`

Prevents keyboard window being continually if text entry area is repeatedly clicked.

`gDDguiCaretColour%`

This sets the default colour of the caret. The default value is 0 (black)

`gDDguiMinControlDimension%`

This sets the minimum height of the caption bar as well as the height of each line in a listbox. The default value is 16

Positioning

Until standard GUI systems, widget positioning is based on whether the current defined widget will fit horizontally – effectively its all based on the width of the previous widget.

If the current widget wont fit horizontally, it will be moved vertically below the last widget.

If it wont fit below the last widget, then a scroll bar on the window will be created and the widget will be added in the newly available area of the window.

Read-Only

Buttons, check boxes and various other widgets can be made read-only or read and writeable.

Read-only mode does not allow the user to change anything about the widget, whilst read and writeable will allow the user to change, select or edit the widget.

Read-only can be enabled or disabled by using DDgui_set with the "READONLY" parameter.

An example for this would be :

```
DDgui_button("t1","moo",100,50)
DDgui_button("t2","moo",100,50)
DDgui_set("t1","READONLY",TRUE)
```

Which makes the "t1" button read-only, which means that if the user moves the mouse over or presses the first button, nothing will happen.

A widget's colour wont change if the mouse is moved over a READ-ONLY widget.

To disable read-only mode, the "READONLY" parameter should be set to **FALSE** :

```
DDgui_set("t1","READONLY",FALSE)
```

Tool-Tips

Tool-tips allows the user to be given succinct information when the user moves over a widget.

A tool-tip is created using :

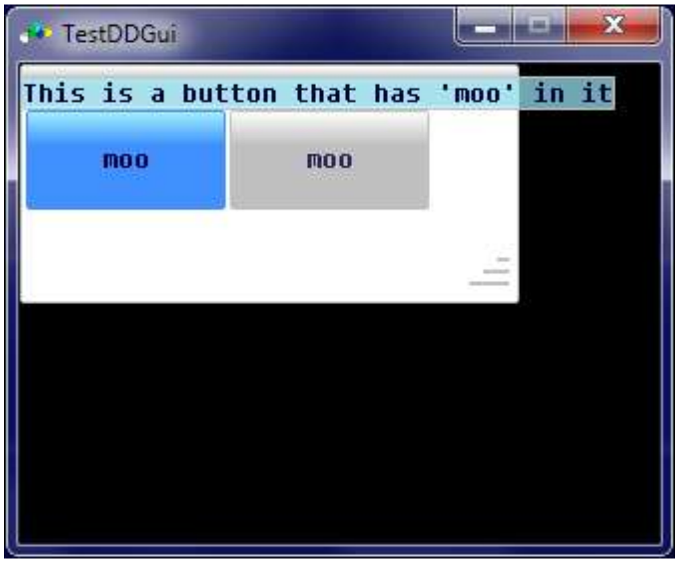
```
DDgui_set(id$, “TOOLTIP”, text$)
```

id$ is the ID of the widget that you want to create a tool-tip for. text$ is tool-tip text that you want to be displayed.

An example of using this is :

```
DDgui_set("t1","TIPTEXT","This is a button that has 'moo' in it")
```

Which looks like :



The tool-tip is displayed until the user moves the mouse away from the widget

Input

An input window (with an on-screen keyboard), can be created to allow easy user-input. It should only be used when all windows have been closed (or haven't been created yet), otherwise you may get a run-time error.

The input window can be created with :

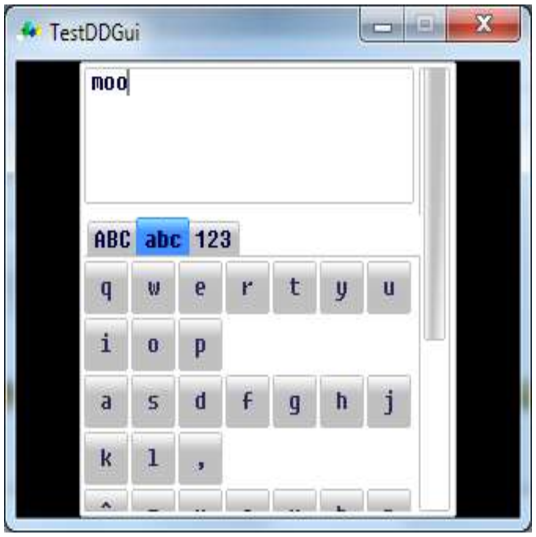
```
result$ = DDgui_input$(text$, specialChars%)
```

text$ is the initial text that will populate the window. **specialChars%**, if set to **TRUE** will enable extended characters to be used. If this variable is set to **FALSE**, then these wont be available.

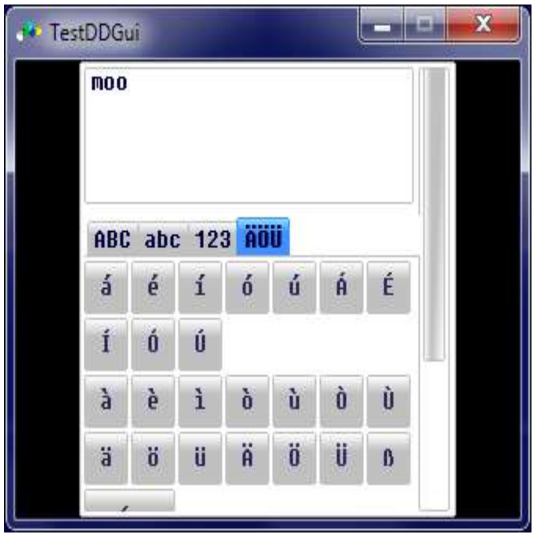
This routine returns all characters entered by the user. If the user has pressed the CANCEL button then the text returned will be the same as that passed originally to this function.

An input window looks like :

Window without special chars available



Window with special chars.



Progress Bar

A progress bar can be activated to show the user that your program hasn't crashed during long operations.

It can be activated and de-activated with :

```
DDgui_WaitCursor%:  bWait% [, iSpriteId%]
```

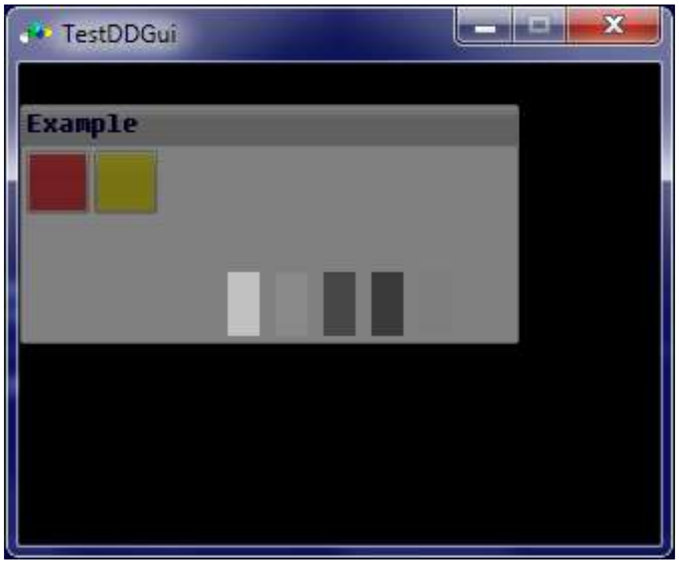
bWait% denotes whether to activate (**bWait%** = **TRUE**) or de-activate (**bWait%** = **FALSE**) the process bar.

iSpriteId%, if present, denotes which sprite index to use to make a copy of the background. If this value is < 0, then a free index will found in order to make the copy.

This command should be used after the `DDgui_show` command. Using it before `DDgui_show` will either produce a nasty flickering effect or a blank background.

You should also note that your loop should NOT call `SHOWSCREEN` as that will be done in this function.

The progress bar looks like :



Error Messages

In debug mode, you could find that you may come across one or more of these errors :

DDgui_get: Widget not found

This occurs when a widget ID has not be found, possibly due to being misspelt.

DDgui_set dialog ("") property: x is unknown

This error occurs when the property “x” is unknown, possibly because it has been misspelt.

Must overwrite: DDgui_handle_user

When creating a custom widget, both DDgui_draw_user% and DDgui_handle_user functions need to be created

DDgui_set and DDgui_get/DDgui_get$ values

DDgui_set values available :

ID Value	Name	Value
“”	“MOVEABLE”	TRUE – Enable window to be moved (and create title bar) FALSE – Disable window movement (and remove title bar)
“” id$	“TEXT”	If this is not an empty string, then a title bar will be created Sets widget to new text
“”	“SCALEABLE”	TRUE – Allows window to be resized FALSE – Disables window re-sizing
“”	“XPOS”	Sets horizontal position of window
“”	“YPOS”	Sets vertical position of window
“” id$	“WIDTH”	Sets width of window Sets width of widget
“” id$	“HEIGHT”	Sets height of window Sets height of window
id$	“ALIGN”	Aligns the widget into the available space : -1 : Align left 0 : Centre 1 : Align right
id$	“TOOLTIP”	Set tool-tip text for widget
id$	“SELECT”	Select which radio button, checkbox or line to be activated
“”	“INKEY”	Adds character to widget when it is updated (like mouse moves over it)
id$	“MINVAL”	Sets minimum value for slider
id$	“MAXVAL”	Sets maximum value for slider
id$	“STEP”	Sets step value for slider

`DDgui_get`/`DDgui_get$` values available :

ID Value	Name	Returns
“”	“MOVEABLE”	0 if window can't be moved Any other value if it can
“”	“SCALABLE”	0 if window can't be resized Any other value if it can
“”	“XPOS”	Horizontal position of window
“”	“YPOS”	Vertical position of window
“” id$	“WIDTH”	Width of window Width of widget
“” id$	“HEIGHT”	Height of window Height of window
id$	“HOVER”	FALSE if mouse is not over widget TRUE if the mouse is over the widget
id$	“CLICKED”	TRUE if user clicked over widget FALSE if user clicked over widget
id$	“READONLY”	0 if widget is not read-only Any other value if it is
id$	“ALIGN”	Alignment of widget -1 : Align left 0 : Centre 1 : Align right
id$	“TOOLTIP”	Current tool-tip text for widget
“” id$	“TEXT”	Window title Text from widget
id$	“SELECT”	Which item (or line) has been selected. Only useful for widgets like radio buttons, check boxes etc.
id$	“COUNT”	How many lines there are in a list box
id$	“SELSTART”	Character position where highlighting starts in text box. If < 0, then no highlighting is taking place
id$	“SELEND”	Character position where highlighting ends in text box. If < 0, then no highlighting is taking place If this value is the same as that returned by “SELSTART”, then no highlighting is taking place
“”	“INKEY”	Currently returns an empty string
id$	“MINVAL”	Returns minimum value for slider
id$	“MAXVAL”	Returns maximum value for slider
id$	“STEP”	Returns step value for slider

Part 2

Function List

DDgui_pushdialog: x%, y%, width%, height% *[, center_to_screen% = FALSE]*

Creates a DDgui window

Example :

```
DDgui_pushdialog(0,0,100,100,TRUE)
```

Result :

This creates a window and centres it to the screen

DDgui_popdialog()

Closes a DDGui window

Example :

```
DDgui_popdialog()
```

Result :

This closes a DDGui window

DDgui_set values available :

None

DDgui_get/DDgui_get$ values available :

None

DDgui_widget%: id$, caption$ *[, width%=0, height%=0]*

Creates a widget

Example :

```
DDgui_widget(“example”,”This is a widget”)
```

Result :

This creates a border-less, non-interactive text

DDgui_button: id$, caption$, *[width%=0, height%=0]*

Creates a button

Example :

```
DDgui_button(“button1”,”Press Me”)
DDgui_button(“button2”,”SPR_B0”,64,64)
```

Result :

This function creates a button with text, a button with a sprite or a filled button, which, when pressed, will open the colour dialog window.

DDgui_slider: id$, value *[, width%=100, height%=16]*

Create a slider

Example :

```
DDgui_slider("slider",0.5,100,20)
```

Result :

This will create a slider 100 pixels wide, with the slider graphic being in the middle.

DDgui_toolbar: id_buttons$[], id_sprites[]

Create a toolbar

Example :

```
LOCAL id$[];  DIMDATA id$[],"test1","test2"
LOCAL sprites[]; DIMDATA sprites[],0,1
DDgui_toolbar(id$[],sprites[])
```

Result :

This creates a list of buttons in one go – saves having to call DDgui_button commands multiple times.

DDgui_checkbox: id$, caption$ *[, width%=0, height%=0]*

Create a check box

Example :

```
DDgui_checkbox("test","This is an example")
```

Result :

This creates a check box, which is an item that has two states : selected or not selected.

DDgui_radio: id$, texts$ *[, width%=0]*

Create a selection where only one can be selected

Example :

```
DDgui_radio("selection","Option 1|Option 2|Option 3")
```

This command creates a group of options, where only one can be selected.

DDgui_file: id$, caption$, filter$ *[, width%=0, height%=0]*

Creates a file dialog button

Example :

```
DDgui_file("open","Select A File","*.*")
```

Result :

This command opens the file requester window (Windows) or calls the INPUT command (Other platforms) when this button is clicked.

See *GLBasic Programmers Reference Guide* for more details on the File Requester and INPUT commands, but be away that the File Requester will crash on Windows 7 x64 in Debug mode.

DDgui_combo: id$, texts$ *[, width%=0, height%=0]*

Display a multiple selection list

Example :

```
DDgui_combo("entry","Level 1|Level 2|Level 3")
```

Result :

An alternative to list boxes, this allows selection of one of a number of items, when the button is pressed

DDgui_list: id$, texts$ *[, width%=0, height%=0]*

Display a list of items

Example :

```
DDgui_list("entry","Level 1|Level 2|Level 3")
```

Result :

Display a list of items, of which one can be selected. The main difference between these is that the list is displayed without the need for the user to click on the selection button.

DDgui_text: id$, text$ *[, width%=0, height%=0]*

Create a text entry widget

Example :

```
DDgui_text("name","Example Name",140)
```

Result :

This widget creates an area where the user can enter text using either the keyboard or an on-screen entry system.

Pressing RETURN will cause the caret to move to the next line, possibly causing the previous text to scroll up.

Pressing the TAB key will generate an space. HOME and END will move the caret to the beginning or end of the current line, whilst the cursor keys will move the caret around one character at a time.

There is no limit to the amount of text that the user can enter.

DDgui_singletext: id$, text$ *[,width%=0]*

Creates a single line text entry widget

Example :

```
DDgui_singletext(“oneline”,”example”)
```

Result :

This function creates a single line text entry widget. With these, pressing TAB will move the caret to the next text or single text entry widget and SHIFT+TAB will move the caret to the previous text or single text entry widget. The next or previous widget being defined by the creation order.

Currently HOME and END have no function

DDgui_tab: id$, captions$

Create a tab list

Example :

```
DDgui_tab("tab","File,name|Options,selection")
DDgui_text("name","Input",100)
DDgui_text("selection","Output",100)
```

Result :

This function allows groups of widgets to be grouped to a related tab, allowing easier viewing.

This command should be used before any other widgets are created, otherwise the display order will be different to what it should be.

Widgets should be created in the order that that are detailed in this command.

`DDgui_framestart:` *[id$="", caption$="", width%=0]*

Create a frame

Example :

```
DDgui_framestart("","Details",100)
     DDgui_text("input","TEXT",90)
DDgui_frameend()
```

Result :

This function allows widgets to be surrounded by a box.

`DDgui_framestart()` should always have a corresponding `DDgui_frameend()` call. If there isn't one, graphical anomalies (like duplicate widgets) will be introduced.

DDgui_frameend()

Finishes a DDgui_framestart() call

Example :

```
DDgui_framestart("","Details",100)
     DDgui_text("input","TEXT",90)
DDgui_frameend()
```

Result :

This command finishes the nearest DDgui_framestart() call, and is always needed.

DDgui_msg%: text$, yes_no% *[, caption$="Information"]*

Display text

Example :

```
IF DDgui_msg("Are you sure that you want to exit ?", _
              TRUE,"Want To Exit ?")
     END
ENDIF
```

Result :

This function allows a brief message to be displayed, with which the user has to press the OK button, or a YES or NO button

DDgui_getitemtext$: id$, index%

Get text from a list-box or combo box

Example :

```
DDgui_list("entry","Level 1|Level 2|Level 3")
DDgui_combo("entry2","Level 1|Level 2|Level 3")

DEBUG DDgui_getitemtext$("entry",2)+"\n"
DEBUG DDgui_getitemtext$("entry2",1)+"\n"
```

Result :

This command allows entry for a given line to be read

DDgui_insertitem%: id$, text$, before_index%

Add text to a list-box or combo box

Example :

```
DDgui_list("entry","Level 1|Level 2|Level 3")
DDgui_combo("entry2","Level 1|Level 2|Level 3")

DDgui_insertitem("entry","Text1",2)
DDgui_insertitem("entry2","Text2",1)
```

Result :

This command allows text to be inserted into a list-box or combo selection at a given position.

DDgui_deleteitem%: id$, index%

Delete a line from a list-box or combo selection

Example :

```
DDgui_list("entry","Level 1|Level 2|Level 3")
DDgui_combo("entry2","Level 1|Level 2|Level 3")

DDgui_deleteitem("entry",2)
DDgui_deleteitem("entry2",1)
```

Result :

This function allows a line (or the complete list) to be deleted from a list-box or combo widget

DDgui_input$: text$ *[, bSpecialChars%=TRUE]*

Allows text entry

Example :

```
DEBUG "Result : "+DDgui_input$("text",FALSE)
```

Result :

This command allows text entry via an external keyboard or the on-screen keyboard, which is automatically displayed.

Pressing the TAB key introduces a space

DDgui_FileDialog$: bOpen% *[, filterstr$="*.*"]*

Opens a file dialog window

Example :

```
DEBUG "Selected : "+DDgui_FileDialog$(TRUE,"*.EXE")
```

Result :

This function opens a file dialog window, in which the list of files in a given directory are filtered according to the contents of **filterstr$**.

You can move down a directory by pressing on ".." and move up by clicking on a directory name.

DDgui_WaitCursor%: bWait% *[, iSpriteId% = -1]*

Create a progress bar

Example :

```
WHILE TRUE
     DDgui_show(FALSE)
     DDgui_WaitCursor(TRUE)
WEND
```

Result :

This command creates (and shows) an animated progress bar. `SHOWSCREEN` should **NOT** be used before or after this command as it will generate a flickering display

DDgui_ColorDlg%: color%

Create a colour picker window

Example :

```
DEBUG "Selected : "+DDgui_ColorDlg(RGB(255,0,0))
```

Result :

This function allows the user to select a colour by tweaking a passed colour, and changing its red, green, blue and brightness levels

DDgui_CenterDialog%

Centres a window

Example :

```
DDgui_pushdialog(0,20,250,100)
DDgui_CenterDialog()
```

Result :

This command centres a window to the current screen resolution.

DDgui_get$: id$, name$

Get data from a widget as a string

Example :

```
DDgui_checkbox("test","This is an example")
DEBUG "Text : "+DDgui_get$("test","TEXT")
```

Result :

This example retrieves specified data from a widget as a string. The type of information received is defined by **name$**

DDgui_get: id$, name$

Get data from a widget as a floating-point value

Example :

```
DDgui_slider("test",0.5,100)
DEBUG "Slider value : "+DDgui_get("test","TEXT")
```

Result :

This example retrieves specified data from a widget as a floating-point value. The type of information received is defined by **name$**

DDgui_set: id$, name$, val$

Set a value in a widget (or window)

Example :

```
DDgui_set("","TEXT","Window Name")
```

Result :

This sets widget (or window) data to a specified value. What the value means is decided by the contents of **name$**

DDgui_UpdateFont: bWantKerning% *[, iPixelsSpaceBetweenLettersIfKerning%=2]*

Enable or disabling proportional fonts

Example :

```
DDgui_UpdateFont(TRUE)
```

Result :

This command enables or disables proportion fonts. These type of fonts look nicer, but as each character can take up a different amount of space, aligning words to a specific position could be a problem.

In V9.052 onwards, the second parameter is not used and has been left in to remain compatible with previous versions of Ddgui.

DDgui_show: only_show_current%

Show DDGui window and contents

Example :

```
DDgui_show(FALSE)
```

Result :

This command can either show all DDgui windows, or the just the one that was last created.

DDgui_resizedialog%: x%,y% *[,width%=0,height%=0]*

Resize a window

Example :

```
DDgui_resizedialog(0,0,40,40)
```

Result :

This command repositions and resizes a window to the specified coordinates and size. All widgets within the window will be re-positioned if needed.

DDgui_hide%: id$ *[, bHide%=TRUE]*

Show or hide a widget

Example :

```
DDgui_slider("test",0.5,100)
DDgui_hide("test",TRUE)
```

Result :

This function hides or shows a widget.

DDgui_userdefined: id$, user_type$, width%, height%

Create a user-defined widget

Example :

```
DDgui_userdefined("test","MOO",100,100)
```

Result :

Creates a user-defined widget, which can be used to do something not support by DDGui

DDgui_spacer: *[width%=10000, height%=0]*

Add space after a widget

Example :

```
DDgui_spacer(5000)
```

Result :

Adds spaces after a widget. It has no effect if no widgets have been created.

DDgui_advancefocus%: *[direction% = 1]*

Move to the next or previous text or single text widget

Example :

```
Ddgui_advancefocus()
```

Result :

Move the cursor to the next text or single text widget

DDgui_setfocus%: id$

Set caret to a text or single text widget

Example :

```
DDgui_setfocus(“entry”)
```

Result :

Set the caret on the text or single text widget

Part 3

Internationalisation

Introduction

If you are displaying text, it would be good if you design your program so that all text can be internationalised.

This means that instead of using text that can't be changed to accommodate other languages, you design your program to use tokens, which will then display text based on the users language settings.

An example use could be something like :

```
DDgui_widget("hello",language$("{{hello}}"),0,0)
```

with the function **language$** taking the token string "*{{hello}}*" and converting it to the word "hello" in English, French, German or any other language.

Thus, I present the following internationalisation routine. It was converted from another BASIC language, and is designed to convert tokens to text.

Additional language files would go in Media/Language/x/LANGUAGE.INI, where "x" is the LOCALE string (returned from PLATFORMINFO("LOCALE")

```
REQUIRE "defaultLanguage.c"

INLINE
	}
	typedef int size_t;

	extern char *defaultLanguage[];
	extern "C" size_t strlen(const char *s);

	namespace __GLBASIC__ {
ENDINLINE

CONSTANT LANGUAGE_ACTIVE%	=	1

TYPE TLocalisation
	flags%
	languageDef$
	languageVer$
	languageAuth$
	map AS TMap

	SECTION_NAME$	=	"LanguageDefinition"

	FUNCTION Initialise%:flags%=0
		IF self.map.Initialise()
			self.flags%=flags%
			self.languageDef$=""
			self.languageVer$=""
			self.languageAuth$=""
			RETURN TRUE
		ELSE
			RETURN FALSE
		ENDIF
	ENDFUNCTION

	FUNCTION Destroy%:
		self.map.Destroy()
	ENDFUNCTION

	FUNCTION SetLanguageDetails%:langDef$,langVer$,langAuth$
		self.languageDef$=langDef$
		self.languageVer$=langVer$
		self.languageAuth$=langAuth$
	ENDFUNCTION

	FUNCTION AddLanguage%:key$,value$
		self.map.Add(key$,value$)
	ENDFUNCTION

	FUNCTION LoadLanguage%:override$=""
	LOCAL path$,language$,temp$
	LOCAL loop%
	LOCAL split$[]

		path$="Media/Language/"
		IF override$=""
			language$=PLATFORMINFO$("LOCALE")
		ELSE
			language$=override$
		ENDIF

		path$=path$+language$+"/LANGUAGE.INI"

		IF DOESFILEEXIST(path$)
			INIOPEN path$

			self.languageDef$=INIGET$(self.SECTION_NAME$,"LanguageID")
			self.languageVer$=INIGET$(self.SECTION_NAME$,"LanguageVersion")
			self.languageAuth$=INIGET$(self.SECTION_NAME$,"LanguageAuthor")

			loop%=0
			temp$=INIGET$(self.SECTION_NAME$,loop%,"")
			DEBUG "T:"+temp$+"\n"
			KEYWAIT
			WHILE temp$<>""
				DIM split$[0]
				IF SPLITSTR(temp$,split$[],",")>=2
					self.map.Add(split$[0],split$[1])
				ENDIF

				INC loop%
				temp$=INIGET$(self.SECTION_NAME$,loop%,"")
			WEND

			INIOPEN ""
			RETURN TRUE
		ELSE
			RETURN FALSE
		ENDIF
	ENDFUNCTION

	FUNCTION GetLanguageID$:
		RETURN self.languageDef$
	ENDFUNCTION

	FUNCTION GetLanguageVersion$:
		RETURN self.languageVer$
	ENDFUNCTION

	FUNCTION GetLanguageAuthor$:
		RETURN self.languageAuth$
	ENDFUNCTION

	FUNCTION LocaliseText$:string$
	LOCAL loop%
	LOCAL one$,tmpPrevChar$,tmpText$
	LOCAL tmpCount%
	LOCAL tmpOpening%[]

		IF bAND(self.flags%,LANGUAGE_ACTIVE%)
			DIM tmpOpening%[LEN(string$)/2]
			FOR loop%=0 TO BOUNDS(tmpOpening%[],0)-1;   tmpOpening%[loop%]=0; NEXT

			loop%=0
			tmpPrevChar$=""
			tmpCount%=0

			WHILE loop%<LEN(string$)
```

```
                                        one$=MID$(string$,loop%,1)
                                        SELECT one$
                                                CASE    "{"     // Start of token
                                                                        IF tmpPrevChar$ = one$
                                                                                tmpOpening%[tmpCount%] = loop%+1
                                                                                INC tmpCount%
                                                                                tmpPrevChar$=""
                                                                        ELSE
                                                                                tmpPrevChar$=one$
                                                                        ENDIF

                                                CASE    "}"     // End of token
                                                                        IF tmpPrevChar$ = one$
                                                                                DEC tmpCount%

        tmpText$=MID$(string$,tmpOpening%[tmpCount%],loop%-tmpOpening%[tmpCount%]-1)
                                                                                SELECT UCASE$(tmpText$)
                                                                                        CASE    "DOCS"

        tmpText$=PLATFORMINFO$("DOCUMENTS")

                                                                                        CASE    "WHAT"

        tmpText$=PLATFORMINFO$("")

                                                                                        CASE    "ID"

        tmpText$=PLATFORMINFO$("ID")

                                                                                        CASE    "DEVICE"

        tmpText$=PLATFORMINFO$("DEVICE")

                                                                                        CASE    "COMPILED"

        tmpText$=PLATFORMINFO$("COMPILED")

                                                                                        DEFAULT

        DEBUG "Tmp : "+tmpText$+"\n"

        tmpText$ = self.LocaliseText$(self.map.GetValue$(tmpText$))

                                                                                ENDSELECT


        string$=MID$(string$,0,tmpOpening%[tmpCount%]-2)+tmpText$+MID$(string$,loop%+1)
                                                                                INC loop%,LEN(tmpText$)-(loop%+3-
tmpOpening%[tmpCount%])

                                                                                tmpPrevChar$=""
                                                                        ELSE
                                                                                tmpPrevChar$=one$
                                                                        ENDIF

                                                DEFAULT
                                                                        tmpPrevChar$=""
                                        ENDSELECT

                                        INC loop%
                                WEND
                        ENDIF

                        RETURN string$
        ENDFUNCTION
ENDTYPE

FUNCTION Localise_TextHelper%:BYREF index%,BYREF token$,BYREF value$
        token$=""
        value$=""

INLINE
        if ((defaultLanguage[index]) && (strlen(defaultLanguage[index])>0))
        {
                token_Str=DGStr(defaultLanguage[(int) index]);
                INC(index);
                if ((defaultLanguage[(int) index]) && (strlen(defaultLanguage[index])>0))
                {
                        value_Str=DGStr(defaultLanguage[(int) index]);
                        INC(index);
                }

                return (DGNat) TRUE;
        }

        return (DGNat) FALSE;
ENDINLINE
ENDFUNCTION

//! Use default text if locale isn't available
FUNCTION Localise_UseDefault%:localise AS TLocalisation
LOCAL result%,index%,token$,value$

        IF localise.LoadLanguage()=FALSE
                index%=0
                result%=Localise_TextHelper(index%,token$,value$)
                WHILE result%=TRUE
                        localise.AddLanguage(token$,value$)
                        result%=Localise_TextHelper(index%,token$,value$)
                WEND

                localise.SetLanguageDetails("English (British)","1.0.0.0","Default Language")
                RETURN FALSE
        ELSE
                RETURN TRUE
        ENDIF
ENDFUNCTION
```

An example of use would be :

```
LOCAL localise as Tlocalise

IF localise.Initialise(LANGUAGE_ACTIVE%)=FALSE
        DEBUG "Unable to initialise TLocalisation"
        RETURN FALSE
ENDIF

localise_UseDefault(localise)
DEBUG “Result : “+localise.LocaliseText$(“{{hello}}”)
```

As it stands, the routine could not be compiled as a default language string is need.

First, you need the Tmap routine. This is covered in slightly more detail in the *GLBasic Programmers Reference Guide*, but is included here for completeness :

```
// --------------------------------- //
// Project: ddd
// Start: Tuesday, October 05, 2010
// IDE Version: 8.120

TYPE tKeyValue
        key$
        value$
ENDTYPE

TYPE TMap
        list[] AS tKeyValue

        //! Initialise type
        //\param None
        //\return TRUE
        FUNCTION Initialise%:
                DIM self.list[0]
                RETURN TRUE
        ENDFUNCTION

        //! Destroy type
        //\param None
        //\return TRUE
        FUNCTION Destroy%:
```

```
		DIM self.list[0]
		RETURN TRUE
	ENDFUNCTION

	//! Add a key and value to the internal array, and sort array afterwards
	//\param key$ - Key to add
	//\param value$ - Value of key
	//\return TRUE if the key and value has been added, FALSE otherwise
	FUNCTION Add%:key$,value$
	LOCAL temp AS tKeyValue

		IF self.search(key$)<0
			// Found
			temp.key$=key$
			temp.value$=value$
			DIMPUSH self.list[],temp
			SORTARRAY self.list[],0
			RETURN TRUE
		ENDIF

		RETURN FALSE
	ENDFUNCTION

	FUNCTION search%:key$
	LOCAL up%,down%,mid%

		up%=0
		down%=BOUNDS(self.list[],0)-1
		WHILE up%<down%
			mid%=(up%+down%)/2
			IF self.list[mid%].key$>key$
				down%=MAX(mid%-1,up%)
			ELSEIF self.list[mid%].key$<key$
				up%=MIN(down%,mid%+1)
			ELSE
				RETURN mid% // Found
			ENDIF
		WEND

		IF BOUNDS(self.list[],0)>0 AND self.list[up%].key$=key$
			RETURN up%
		ELSE
			RETURN -1
		ENDIF
	ENDFUNCTION

	FUNCTION Debug%:
	LOCAL loop AS tKeyValue

		FOREACH loop IN self.list[]
			DEBUG "Key : "+loop.key$+" Value : "+loop.value$+"\n"
		NEXT
		DEBUG "Number of keys and values : "+BOUNDS(self.list[],0)+"\n"
	ENDFUNCTION

	FUNCTION GetValue$:key$,notFound$="NOT_FOUND"
	LOCAL index%

		index%=self.search(key$)
		IF index%>=0
			RETURN self.list[index%].value$
		ELSE
			RETURN notFound$
		ENDIF
	ENDFUNCTION

	FUNCTION DeleteKey%:key$
	LOCAL index%

		index%=self.search(key$)
		IF index%>=0
			DIMDEL self.list[],index%
			RETURN TRUE
		ELSE
			RETURN FALSE
		ENDIF
	ENDFUNCTION
```

```
        FUNCTION Count%:
                RETURN BOUNDS(self.list[],0)
        ENDFUNCTION
ENDTYPE
```

Usually, the text would be added during compilation. Its a simple procedure to add to your programs – save the following as a .c file into the same place as your project (calling it defaultLanguage.c) :

```
char *defaultLanguage[]={
        "startgame",            "Start Game",
        "continuegame",         "Continue Game",
        "options",              "Options",
        "instructions",         "Instructions",
        "hiscores",             "Hiscores",
        "quitgame",             "Quit Game",
        "arcadegame",           "Player V Computer Game",
        "standardgame",         "Standard Game",
        "upto3",                "Up To 3 Wins",
        "prev",                 "Previous Menu",
        "\0",                   "\0"
        };
```

And now, if you use “*{{startgame}}*”, you could get “Start Game” as the resulting string. If you had a French language file setup, “*{{startgame}}*” could return “Commencez le jeu”.

Part 4

DDgui Examples (all of these require DDgui to be added to the project)

Here are some complete programs that use Ddgui for their interface systems :

Graphic To Data

```
// --------------------------------- //
// Project: GraphicToDATA
// Start: Friday, July 09, 2010
// IDE Version: 8.036

CONSTANT fileToLoadWidget$    =      "fileToLoadWidget"
CONSTANT fileToLoad$          =      "fileToLoad"
CONSTANT browseButton$        =      "browse"
CONSTANT addButton$                  =      "add"
CONSTANT listOfFilesList$     =      "filesList"
CONSTANT removeFileButton$    =      "removeFile"
CONSTANT labelWidget$         =      "labelWidget"
CONSTANT labelText$                  =      "labelText"
CONSTANT processButton$              =      "process"
CONSTANT processResult$              =      "processResult"
CONSTANT space$                             =      "    "
CONSTANT VERSION$                    =      "0.0.0.5"

LOCAL sW%,sH%,text$,label$

GETSCREENSIZE sW%,sH%

DDgui_pushdialog(0,0,sW%,sH%)
DDgui_set("","TEXT","Graphics To DATA ("+VERSION$+")")
DDgui_widget(fileToLoadWidget$,"File To Load :",0,0)
DDgui_text(fileToLoad$,"",sW%-100,0)
DDgui_button(browseButton$,"Browse",0,0)
DDgui_button(addButton$,"Add",0,0)
DDgui_list(listOfFilesList$,"",sW%-12,sH%-108)
DDgui_button(removeFileButton$,"Remove File",0,0)
DDgui_widget(labelWidget$,"Label Name :",0,0)
DDgui_text(labelText$,"graphic",sW%-280,0)
DDgui_button(processButton$,"Process",0,0)
DDgui_widget(processResult$,"",sW%-8,0)

WHILE TRUE
       DDgui_show(FALSE)

       IF DDgui_get(browseButton$,"CLICKED")
              text$=DDgui_FileDialog$(TRUE,"*.*")
              IF text$<>"" THEN addFile(listOfFilesList$,text$)
       ELSEIF DDgui_get(addButton$,"CLICKED")
              text$=DDgui_get$(fileToLoad$,"TEXT")
              IF text$<>"" THEN addFile(listOfFilesList$,text$)
       ELSEIF DDgui_get(processButton$,"CLICKED")
              label$=DDgui_get$(labelText$,"TEXT")
              IF label$=""
                     DDgui_msg("Please supply a valid label name",FALSE)
              ELSE
                     processFiles(listOfFilesList$,label$)
              ENDIF
       ELSEIF DDgui_get(removeFileButton$,"CLICKED")
              removeLine(listOfFilesList$)
       ENDIF

       SHOWSCREEN
WEND

FUNCTION addFile%:widget$,text$
       IF DOESFILEEXIST(text$)
              DDgui_set(widget$,"TEXT",DDgui_get$(widget$,"TEXT")+"|"+text$)
       ELSE
              DDgui_msg("The file doesn't exist",FALSE)
       ENDIF
ENDFUNCTION

FUNCTION removeLine%:widget$
LOCAL index%,loop%,text$
LOCAL list$[]
```

```
	index%=DDgui_get(widget$,"SELECT")
	IF index%>=0
		IF DDgui_msg("Are you sure that you want to delete this line ?",TRUE)
			DIM list$[0]
			IF SPLITSTR(DDgui_get$(widget$,"TEXT"),list$[],"|")>0
				text$=""
				FOR loop%=0 TO BOUNDS(list$[],0)-1
					IF loop%<>index%
						text$=text$+list$[loop%]+"|"
					ENDIF
				NEXT
				DDgui_set(widget$,"TEXT",LEFT$(text$,LEN(text$)-1))
			ELSE
				DDgui_msg("There is nothing to split for some reason!",FALSE)
			ENDIF
		ENDIF
	ELSE
		DDgui_msg("There is nothing to select",FALSE)
	ENDIF
ENDFUNCTION

FUNCTION processFiles%:widget$,label$
LOCAL files$[]
LOCAL data%[]
LOCAL fileLoop$,ext$,count%,found%,extLoop%
LOCAL handle%
LOCAL outputFile$
LOCAL maxDataPerLine%=32
LOCAL sW%,sH%,sprite%
LOCAL error$

	handle%=-1
	TRY
		DIM files$[0]
		IF SPLITSTR(DDgui_get$(widget$,"TEXT"),files$[],"|")>0
			// Get the output filename
			outputFile$=DDgui_FileDialog$(FALSE,"*.gbas")
			IF outputFile$="" THEN RETURN FALSE
			IF DOESFILEEXIST(outputFile$)
				IF DDgui_msg("The file already exists.  Do you want to overwrite it
?",TRUE)
					KILLFILE outputFile$
					IF DOESFILEEXIST(outputFile$)
						DDgui_msg("The file could not be deleted - it must be in
use by a program",FALSE)
						RETURN FALSE
					ENDIF
				ELSE
					RETURN FALSE
				ENDIF
			ENDIF

			count%=1
			handle%=GENFILE()
			IF handle%>=0
				IF OPENFILE(handle%,outputFile$,0)
					FOREACH fileLoop$ IN files$[]
						// Validate the extension
						found%=FALSE
						FOR extLoop%=LEN(fileLoop$)-1 TO 0 STEP -1
							IF MID$(fileLoop$,extLoop%,1)="."
								ext$=UCASE$(MID$(fileLoop$,extLoop%+1))
								IF ext$="JPEG" OR ext$="JPG"
									DDgui_msg("JPEG files are not
currently supported.  File will be skipped",FALSE)
								ELSEIF ext$="PNG" OR ext$="BMP"
									found%=TRUE
									BREAK
								ENDIF
							ENDIF
						NEXT

						IF found%=FALSE
							DDgui_msg("The file type is not valid - only PNG
and BMP files are allowed.  File will be skipped",FALSE)
						ENDIF
```

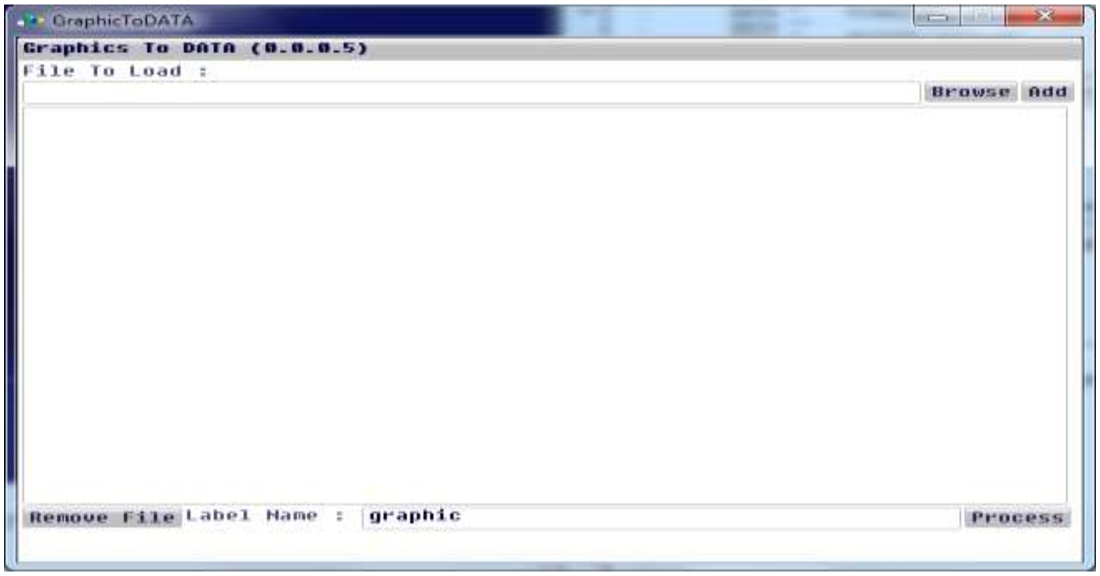
```
                                        IF found%=TRUE
                                             IF DOESFILEEXIST(fileLoop$)
                                                  sprite%=GENSPRITE()
                                                  IF sprite%>=0

     DDgui_set(processResult$,"TEXT","Loading : "+fileLoop$);   DDgui_show(FALSE);    SHOWSCREEN

                                                       LOADSPRITE fileLoop$,sprite%
                                                       GETSPRITESIZE sprite%,sW%,sH%
                                                       IF sW%>0 AND sH%>0
                                                            DIM data%[0]
                                                            IF
SPRITE2MEM(data%[],sprite%)
                                                                 // Only write the
GLBasic code once at the start

     processData(handle%,data%[],maxDataPerLine%,fileLoop$,count%,sW%,sH%,label$,processResult$)

                                                                 IF
count%<BOUNDS(files$[],0)
                                                                      WRITELINE
handle,"\n"
                                                                 ENDIF

                                                                 INC count%
                                                                 LOADSPRITE "",sprite%
                                                            ELSE
                                                                 THROW "Unable to
convert a sprite to data"
                                                            ENDIF
                                                       ELSE
                                                            THROW "Sprite size is
invalid"
                                                       ENDIF
                                                  ELSE
                                                       THROW "Unable to get a sprite handle"
                                                  ENDIF
                                             ELSE
                                                  THROW "A file doesn't exist"
                                             ENDIF
                                        ENDIF
                                   NEXT

                                   CLOSEFILE handle%
                                   DDgui_set(processResult$,"TEXT","Finished");DDgui_show(FALSE);
     SHOWSCREEN
                              ELSE
                                   THROW "Unable to open "+outputFile$
                              ENDIF
                         ENDIF
               ELSE
                    THROW "There is nothing to process"
               ENDIF
     CATCH error$
               IF handle%>=0 THEN CLOSEFILE handle%
               DDgui_msg(error$,FALSE)
               RETURN FALSE
     FINALLY

     RETURN TRUE
ENDFUNCTION

//! Perform RLE compression on data
FUNCTION
processData%:handle%,data%[],maxDataPerLine%,fileName$,count%,sprWidth%,sprHeight%,label$,processResult
$
LOCAL loop%,loop2%,found%
LOCAL v1%,v2%,countDup%,numPerLine%,written%
LOCAL text$
LOCAL graphicDataHexSize%    =    8    // 4 bytes
LOCAL sizeDataHexSize%       =    4    // 2 bytes

     text$=""
     loop%=0
     numPerLine%=0
     written%=FALSE
     WHILE loop%<BOUNDS(data%[],0)
```

```
                v1%=data%[loop%]
                countDup%=1
                INC loop%
                WHILE loop%<BOUNDS(data%[],0) AND countDup%<65535 AND data%[loop%]=v1%
                        IF MOD(loop%,4096)=0
                                DDgui_set(processResult$,"TEXT",loop%+"/"+countDup%);
        DDgui_show(FALSE);      SHOWSCREEN
                        ENDIF

                        INC loop%
                        INC countDup%
                WEND

                IF countDup%>1
                        text$=text$+"*"+decToHex$(countDup%,sizeDataHexSize%)
                ENDIF

                text$=text$+decToHex$(v1%,graphicDataHexSize%)

                INC numPerLine%
                IF numPerLine%>maxDataPerLine%
                        IF written%=FALSE
                                DDgui_set(processResult$,"TEXT","Writing decompression routine");
DDgui_show(FALSE); SHOWSCREEN

                                IF count%=1
                                        writeGLBasicCode(handle)
                                ENDIF

                                DDgui_set(processResult$,"TEXT","Writing DATA start"); DDgui_show(FALSE);
SHOWSCREEN
                                WRITELINE handle%,"// This is the data for the file "+fileName$
                                WRITELINE handle%,"STARTDATA "+label$+"_"+count%+":"
                                WRITELINE handle%,space$+"DATA "+sprWidth%+","+sprHeight%
                        ENDIF

                        WRITELINE handle%,space$+"DATA "+CHR$(34)+text$+CHR$(34)

                        written%=TRUE
                        text$=""
                        numPerLine%=0
                ENDIF
        WEND

        // Is there anything left over ?
        IF text$<>""
                DDgui_set(processResult$,"TEXT","Writing left-over text"); DDgui_show(FALSE); SHOWSCREEN
                WRITELINE handle%,space$+"DATA "+CHR$(34)+text$+CHR$(34)
                written%=TRUE
        ENDIF

        IF written%=TRUE
                DDgui_set(processResult$,"TEXT","Writing terminator"); DDgui_show(FALSE); SHOWSCREEN
                WRITELINE handle%,space$+"DATA "+CHR$(34)+"--------"+CHR$(34)
                WRITELINE handle%,"ENDDATA"
        ENDIF
ENDFUNCTION

// This is the function that creates the function for reading in the data
FUNCTION writeGLBasicCode%:handle%
LOCAL text$

        RESTORE readSpriteData
        READ text$
        WHILE text$<>"*"
                text$=REPLACE$(text$,"'",CHR$(34))
                text$=REPLACE$(text$,"@","\\")
                DDgui_WaitCursor(TRUE)
                WRITELINE handle%,text$
                READ text$
        WEND
ENDFUNCTION


FUNCTION decToHex$:value%,length%=4
        IF length%<=0
                RETURN "0"
```

```
	ENDIF

INLINE
	DGStr result;
	unsigned long value2;
	int temp,digit;

		value2=(unsigned long) value;
		result=DGStr("");

		for (digit=length; digit>=1; digit--)
		{
			temp=value2 % 16;
			if (temp<10)
			{
				result=CHR_Str(temp+48)+result;
			}
			else
			{
				result=CHR_Str((temp-10)+65)+result;
			}

			value2/=16;
		}

		return result;
ENDINLINE
ENDFUNCTION

STARTDATA readSpriteData:
	DATA "//! This function will create a sprite from lines of hexadecimal pixel data.  You MUST set
RESTORE to the appropriate data statement first!"
	DATA "//@param  sprite% - The sprite number which will be created.  If this is < 0 then a sprite
handle will be automatically found if possible"
	DATA "//@return >=0 if the function completed successfully.  The value returned is the sprite
number used."
	DATA "FUNCTION readSpriteData%:sprite%=-1"
	DATA "LOCAL text$,error$,one$,loop%,index%"
	DATA "LOCAL array%[]"
	DATA "LOCAL textLen%  =      32"
	DATA "LOCAL hexSize%  =      8"
	DATA "LOCAL dupSize%  =   4"
	DATA "LOCAL width%,height%"
	DATA ""
	DATA "    TRY"
	DATA "        IF sprite%<0"
	DATA "            sprite%=GENSPRITE()"
	DATA "            IF sprite%<0 THEN THROW 'Unable to get a sprite handle'"
	DATA "        ENDIF"
	DATA ""
	DATA "        READ width%,height%"
	DATA "        IF width%<=0 OR height%<=0 THEN THROW 'Sprite width and/or height is invalid'"
	DATA "        IF width%*height%<0 THEN THROW 'Sprite width and/or height is too large'"
	DATA ""
	DATA "        DIM array%[width%*height%]"
	DATA "        index%=0"
	DATA ""
	DATA "        READ text$"
	DATA "        WHILE text$<>'--------'"
	DATA "             IF LEN(text$)<8 THEN THROW 'A line has an invalid number of characters'"
	DATA ""
	DATA "            loop%=0"
	DATA "            WHILE loop%<LEN(text$)"
	DATA "               IF MID$(text$,loop%,1)='*'"
	DATA "                  LOCAL value%,amount%"
	DATA ""
	DATA "                  // Duplicate data"
	DATA "                  amount%=hexToDec(MID$(text$,loop%+1,dupSize%))"
	DATA "                  value%=hexToDec(MID$(text$,loop%+1+dupSize%,hexSize%))"
	DATA "                  WHILE amount%>0"
	DATA "                      array%[index%]=value%"
	DATA "                      INC index%"
	DATA "                      DEC amount%"
	DATA "                  WEND"
	DATA ""
	DATA "                  INC loop%,hexSize%+dupSize%+1"
	DATA "               ELSE"
```

```
        DATA "                    array%[index%]=hexToDec(MID$(text$,loop%,hexSize%))"
        DATA "                    INC loop%,hexSize%"
        DATA "                    INC index%"
        DATA "                ENDIF"
        DATA "             WEND"
        DATA "             READ text$"
        DATA "             SLEEP 1"
        DATA "         WEND"
        DATA ""
        DATA "         // All finished, so we now call the MEM2SPRITE function"
        DATA "         IF MEM2SPRITE(array%[],sprite%,width%,height%)=FALSE THEN THROW 'Unable TO convert
DATA TO a sprite'"
        DATA "     CATCH error$"
        DATA "         // Put in your own code to display errors here.  Uncomment to use DDGui"
        DATA "         // DDgui_msg(error$,FALSE)"
        DATA "         // You MUST keep the following command"
        DATA "         RETURN -1"
        DATA "     FINALLY"
        DATA ""
        DATA "     RETURN sprite%"
        DATA "ENDFUNCTION"
        DATA ""
        DATA "//! Convert a hexadecimal number of an integer"
        DATA "//param hex$ - Hexidecimal string"
        DATA "//@return The decimal value of the passed string"
        DATA "FUNCTION hexToDec%:hex$"
        DATA "LOCAL i%"
        DATA "LOCAL j%"
        DATA "LOCAL loop%"
        DATA ""
        DATA "     i%=0"
        DATA "     j%=0"
        DATA "     FOR loop%=0 TO LEN(hex$)-1"
        DATA "         i%=ASC(MID$(hex$,loop%,1))-48"
        DATA "         IF 9<i% THEN DEC i%,7"
        DATA "         j%=bOR(j%*16,bAND(i,15))"
        DATA "     NEXT"
        DATA ""
        DATA "     RETURN j%"
        DATA "ENDFUNCTION"
        DATA ""
        DATA "*"
ENDDATA
```

This program converts one or more sprites to compressed hexadecimal data, and looks like :



Vector Editor

```
// -------------------------------- //
// Project: VectorEditor
// Start: Saturday, July 17, 2010
// IDE Version: 8.036

// Working mode constants
CONSTANT VERSION$           =      "1.0.0.2"

CONSTANT MODE_ADD%          =      0
CONSTANT MODE_MOVE%         =      1
CONSTANT MODE_REMOVE% =      2
CONSTANT MODE_INSERT% =      3

CONSTANT CELL_NODE%         =      0

CONSTANT NODE_HANDLE% =      -1
CONSTANT NODE_NODE1%  =      0
CONSTANT NODE_NODE2%  =      1
CONSTANT NODE_NODE3%  =      2
CONSTANT NODE_NODE4%  =      3
CONSTANT NODE_NODE5%  =      4
CONSTANT NODE_NODE6%  =      5
CONSTANT NODE_NODE7%  =      6
CONSTANT NODE_NODE8%  =      7
CONSTANT NODE_NODE9%  =      8
CONSTANT NODE_NODE10% =      9
CONSTANT NODE_USER%         =      10

CONSTANT INVALID%           =      -1

CONSTANT SECTION_NAME$=      "OBJECT"
CONSTANT KEY_AUTHOR$  =      "AUTHOR"
CONSTANT KEY_OBJECTNAME$=    "NAME"
CONSTANT KEY_HANDLE$  =      "HANDLE"
CONSTANT KEY_DATA$          =      "DATA"
CONSTANT KEY_INUSE$         =      "INUSE" // Groups in use

CONSTANT SECTION_USER$=      "USER"
CONSTANT KEY_COUNT$         =      "COUNT"

CONSTANT COLOUR_HANDLE1%=    RGB(128,128,128)
CONSTANT COLOUR_HANDLE2%=    RGB(64,64,64)

CONSTANT MAX_NODEGROUPS%=    10
CONSTANT MAX_USERGROUPS%=    10

// Holds X & Y value for a node or user cell
TYPE tNode
       x%                    // X & Y cell position
       y%
       currentColour%        // Colour for node not connecting to first/last node
       lastColour%
ENDTYPE

TYPE tNodeGroup
       name$          // Node name
       nodes[] AS tNode
ENDTYPE

// Holds user cell information when reading in from INI file
TYPE tCellInfo
       name$
       nodeIndex%            // NODE_NODE1-10 for nodes, > NODE_NODE10 for user cells
       currentColour%
       lastColour%
       maxRange%
ENDTYPE

TYPE tFindItem
       groupIndex%    // < 0, if not in use, 0 - 9 for node group, > 9 for user cell
       nodeIndex%     // Index into array
ENDTYPE

TYPE tUndo
```

```
        name$
        nodeGroup[] AS tNodeGroup
        userCells[] AS tNodeGroup
ENDTYPE

// Displays the grid
TYPE TGrid
        x%
        y%
        gridSprite%
        GRID_WIDTH%     =       33
        GRID_HEIGHT%    =       33
        cellWidth%
        cellHeight%
        halfCellWidth%
        halfCellHeight%
        totalHeight%
        totalWidth%
        handleX%
        handleY%
        showX%
        showY%

        nodeGroup[] AS tNodeGroup
        userCells[] AS tNodeGroup
        undo[] AS tUndo

        actionType%                         // Whether we're in add, move, remove or insert mode
        cellMode AS tCellInfo // Type of cell information to process - add a node or user-defined cell
information

        leftButtonPressed%
        findItem AS tFindItem

        FUNCTION Initialise%:screenWidth%,screenHeight%
        LOCAL screen%=0
        LOCAL lx%,ly%

                self.gridSprite%=GENSPRITE()
                IF self.gridSprite%<0 THEN RETURN FALSE

                DIM self.userCells[MAX_USERGROUPS%]
                FOR ly%=0 TO MAX_USERGROUPS%-1
                        self.userCells[ly%].name$="User Cell #"+ly%
                NEXT

                DIM self.nodeGroup[MAX_NODEGROUPS%]
                FOR ly%=0 TO MAX_NODEGROUPS%-1
                        self.nodeGroup[ly%].name$="Node Group #"+ly%
                NEXT

                self.ClearData()

                self.cellWidth%=INTEGER((screenWidth%/2)/self.GRID_WIDTH%)
                self.cellHeight%=self.cellWidth%
                self.halfCellWidth%=INTEGER(self.cellWidth%/2)
                self.halfCellHeight%=self.halfCellWidth%

                self.totalWidth%=self.cellWidth%*self.GRID_WIDTH%
                self.totalHeight%=self.cellHeight%*self.GRID_HEIGHT%

                self.x%=((screenWidth%/2)-self.totalWidth%)/2
                self.y%=(screenHeight%-self.totalHeight%)/2

                self.showX%=(screenWidth%/2)+(((screenWidth%/2)-self.GRID_WIDTH%)/2)
                self.showY%=screenHeight%-4-self.GRID_HEIGHT%

                CREATESCREEN screen%,self.gridSprite%,self.totalWidth%+2,self.totalHeight%+2
                USESCREEN screen%

                FOR ly%=1 TO self.GRID_HEIGHT%-1
                        DRAWLINE
1,ly%*self.cellHeight%,self.totalWidth%,ly%*self.cellHeight%,RGB(0,0,200)
                NEXT

                FOR lx%=1 TO self.GRID_WIDTH%-1
                        DRAWLINE lx%*self.cellWidth%,1,lx%*self.cellWidth%,self.totalHeight%,RGB(0,0,200)
                NEXT

                DRAWLINE 0,0,self.totalWidth%,0,RGB(0,0,200)
                DRAWLINE self.totalWidth%+1,0,self.totalWidth%+1,self.totalHeight%+1,RGB(0,0,200)
                DRAWLINE 0,self.totalHeight%+1,self.totalWidth%+1,self.totalHeight%+1,RGB(0,0,200)
                DRAWLINE 0,0,0,self.totalHeight%+1,RGB(0,0,200)

                USESCREEN -1

                self.handleX%=INTEGER(self.GRID_WIDTH%/2)
                self.handleY%=INTEGER(self.GRID_HEIGHT%/2)

                self.leftButtonPressed%=FALSE
                self.findItem.groupIndex%=INVALID%
                self.findItem.nodeIndex%=INVALID%

                RETURN TRUE
        ENDFUNCTION

        //! Change action type
        FUNCTION SetActionType%:actionType%
                self.actionType%=actionType%
        ENDFUNCTION

        FUNCTION SetNodeInfo%:cellInfo AS tCellInfo
                self.cellMode=cellInfo
        ENDFUNCTION

        //! Display the grid, nodes and user cells
        FUNCTION DisplayGrid%:
        LOCAL loop AS tNode
        LOCAL loop2%,loop3%,size%,x1%,y1%,x2%,y2%,nextIndex%,colour%

                // Display the grid first
                ALPHAMODE 0.0
                DRAWSPRITE self.gridSprite%,self.x%,self.y%

                // Draw the handle
                FOR x1%=0 TO self.GRID_WIDTH%-1
                        self.Rect(x1%,self.handleY%,COLOUR_HANDLE2%)
                NEXT

                FOR y1%=0 TO self.GRID_HEIGHT%-1
                        self.Rect(self.handleX%,y1%,COLOUR_HANDLE2%)
                NEXT

                self.Rect(self.handleX%,self.handleY%,COLOUR_HANDLE1%)

                // Draw user-defined cells
                FOR loop2%=0 TO MAX_NODEGROUPS%-1
                        FOREACH loop IN self.userCells[loop2%].nodes[]
                                self.Rect(loop.x%,loop.y%,loop.currentColour%)
                        NEXT
                NEXT

                // Draw node cells
                FOR loop2%=0 TO MAX_NODEGROUPS%-1
                        FOREACH loop IN self.nodeGroup[loop2%].nodes[]
                                self.Rect(loop.x%,loop.y%,loop.currentColour%)
                        NEXT
                NEXT

                // Draw node lines
                FOR loop2%=0 TO MAX_NODEGROUPS%-1
                        size%=BOUNDS(self.nodeGroup[loop2%].nodes[],0)
                        IF size%>1
                                FOR loop3%=0 TO size%-1

        x1%=self.x%+(self.nodeGroup[loop2%].nodes[loop3%].x%*self.cellWidth%)+1+self.halfCellWidth%

        y1%=self.y%+(self.nodeGroup[loop2%].nodes[loop3%].y%*self.cellHeight%)+1+self.halfCellHeight%
                                        nextIndex%=loop3%+1
                                        IF nextIndex%>=size%
                                                nextIndex%=0
                                                colour%=RGB(255,255,0)
                                        ELSE
                                                colour%=self.nodeGroup[loop2%].nodes[loop3%].currentColour%
                                        ENDIF


        x2%=self.x%+(self.nodeGroup[loop2%].nodes[nextIndex%].x%*self.cellWidth%)+1+self.halfCellWidth%

        y2%=self.y%+(self.nodeGroup[loop2%].nodes[nextIndex%].y%*self.cellHeight%)+1+self.halfCellHeight
%
                                        DRAWLINE x1%,y1%,x2%,y2%,colour%

                                        x1%=self.showX%+self.nodeGroup[loop2%].nodes[loop3%].x%
                                        y1%=self.showY%+self.nodeGroup[loop2%].nodes[loop3%].y%
                                        x2%=self.showX%+self.nodeGroup[loop2%].nodes[nextIndex%].x%
                                        y2%=self.showY%+self.nodeGroup[loop2%].nodes[nextIndex%].y%
                                        DRAWLINE x1%,y1%,x2%,y2%,RGB(0,0,255)

                                NEXT
                        ENDIF
                NEXT
        ENDFUNCTION

        FUNCTION Rect%:x%,y%,colour%
                DRAWRECT
self.x%+(x%*self.cellWidth%)+1,self.y%+(y%*self.cellHeight%)+1,self.cellWidth%,self.cellHeight%,colour%
        ENDFUNCTION

        FUNCTION Process%:
        LOCAL mx%,my%,b1%,b2%

                MOUSESTATE mx%,my%,b1%,b2%

                IF b1%
                        SELECT self.actionType%
                                CASE    MODE_ADD%
                                                                IF self.leftButtonPressed%=FALSE
                                                                        IF self.InGridArea(mx%,my%)
                                                                                IF
self.cellMode.nodeIndex%>=NODE_NODE1% AND self.cellMode.nodeIndex%<=NODE_NODE10%

        self.AddNodeItem(mx%,my%)
                                                                                ELSE

        self.AddUserItem(mx%,my%)
                                                                                ENDIF


        self.leftButtonPressed%=TRUE
                                                                        ENDIF
                                                                ENDIF

                                CASE    MODE_MOVE%
                                                                // Move a node or user-defined cell
                                                                IF self.leftButtonPressed%=FALSE
                                                                        IF
self.FindNode(mx%,my%,self.findItem)=TRUE

        self.leftButtonPressed%=TRUE

                                                                        ENDIF
                                                                ELSE
                                                                        IF
self.findItem.nodeIndex%<>INVALID%

        self.MoveItem(mx%,my%,self.findItem)

                                                                        ENDIF
                                                                ENDIF

                                CASE    MODE_REMOVE%
                                                                // Remove a node or user-defined cell
                                                                IF self.leftButtonPressed%=FALSE
                                                                        IF
self.FindNode(mx%,my%,self.findItem)=TRUE

        self.leftButtonPressed%=TRUE

                                                                        ENDIF
                                                                ELSE
                                                                        IF
self.findItem.nodeIndex%<>INVALID%
```

```
        self.DeleteItem(self.findItem)

        self.findItem.nodeIndex%=INVALID%
                                                                                ENDIF
                                                                        ENDIF

                                CASE    MODE_INSERT%
                                                                        // Insert and sort - only with nodes
                                                                        IF self.leftButtonPressed%=FALSE
                                                                                IF self.InGridArea(mx%,my%)
                                                                                        IF
self.cellMode.nodeIndex%>=NODE_NODE1% AND self.cellMode.nodeIndex%<=NODE_NODE10%

        self.AddNodeItem(mx%,my%)

                                                                                                FOR mx%=0 TO
BOUNDS(self.nodeGroup[self.cellMode.nodeIndex%-NODE_NODE1%].nodes[],0)-1
                                                                                                        DEBUG

self.nodeGroup[self.cellMode.nodeIndex%-NODE_NODE1%].nodes[mx%].x%+"
"+self.nodeGroup[self.cellMode.nodeIndex%-NODE_NODE1%].nodes[mx%].y%+"\n"
                                                                                                NEXT
                                                                                                DEBUG "\n"
//
                                                                                                // Sort
                                                                                                SORTARRAY
self.nodeGroup[self.cellMode.nodeIndex%-NODE_NODE1%].nodes[],ADDRESSOF(sortRoutine)

                                                                                                FOR mx%=0 TO
BOUNDS(self.nodeGroup[self.cellMode.nodeIndex%-NODE_NODE1%].nodes[],0)-1
                                                                                                        DEBUG

self.nodeGroup[self.cellMode.nodeIndex%-NODE_NODE1%].nodes[mx%].x%+"
"+self.nodeGroup[self.cellMode.nodeIndex%-NODE_NODE1%].nodes[mx%].y%+"\n"
                                                                                                NEXT
                                                                                        ELSE

        self.AddUserItem(mx%,my%)
                                                                                        ENDIF


        self.leftButtonPressed%=TRUE
                                                                                ENDIF
                                                                        ENDIF


                        ENDSELECT
                ELSE
                        self.leftButtonPressed%=FALSE
                        self.findItem.groupIndex%=INVALID%
                        self.findItem.nodeIndex%=INVALID%
                ENDIF

        ENDFUNCTION

        //! Are mouse coordinates withing grid area ?
        FUNCTION InGridArea%:mx%,my%
                IF (mx%>=self.x% AND mx%<=self.x%+self.totalWidth%) AND _
                        (my%>=self.y% AND my%<=self.y%+self.totalHeight%)
                        RETURN TRUE
                ELSE
                        RETURN FALSE
                ENDIF
        ENDFUNCTION

        //! Convert mouse coordinates to cell X & Y values
        FUNCTION MouseXYToCellXY%:mx%,my%,BYREF cellX%,BYREF cellY%
                cellX%=INTEGER((mx%-(self.x%+1))/self.cellWidth%)
                cellY%=INTEGER((my%-(self.y%+1))/self.cellHeight%)
        ENDFUNCTION

        //! Delete a node or user cell
        FUNCTION DeleteItem%:found AS tFindItem
                IF found.groupIndex%>=NODE_NODE1% AND found.groupIndex%<=NODE_NODE10%
                        // Deleting a node
                        DIMDEL self.nodeGroup[found.groupIndex%-NODE_NODE1%].nodes[],found.nodeIndex%
                ELSEIF found.groupIndex%>=NODE_USER%
                        // Deleting User cell
                        DIMDEL self.userCells[found.groupIndex%-NODE_USER%].nodes[],found.nodeIndex%
```

```
		ELSE
			DDgui_msg("Invalid group index value : "+found.groupIndex%,FALSE)
		ENDIF
	ENDFUNCTION

	//! Add a node to the node list into the current node group
	FUNCTION AddNodeItem%:mx%,my%
	LOCAL node AS tNode

		self.AddToUndo("Adding Node")
		self.MouseXYToCellXY(mx%,my%,node.x%,node.y%)
		node.currentColour%=self.cellMode.currentColour%
		node.lastColour%=self.cellMode.lastColour%
		DIMPUSH self.nodeGroup[self.cellMode.nodeIndex%-NODE_NODE1%].nodes[],node
	ENDFUNCTION

	//! Add a user-defined cell to the correct group
	FUNCTION AddUserItem%:mx%,my%
	LOCAL node AS tNode

		IF ((self.cellMode.maxRange%>0) AND (BOUNDS(self.userCells[self.cellMode.nodeIndex%-
NODE_USER%].nodes[],0)<self.cellMode.maxRange%)) OR (self.cellMode.maxRange%<0)
			self.AddToUndo("Adding User Cell")
			self.MouseXYToCellXY(mx%,my%,node.x%,node.y%)
			node.lastColour%=self.cellMode.lastColour%
			node.currentColour%=self.cellMode.currentColour%
			DIMPUSH self.userCells[self.cellMode.nodeIndex%-NODE_USER%].nodes[],node
		ELSE
			DDgui_msg("You are only allow to use "+self.cellMode.maxRange%+" of the selected
cell",FALSE)
		ENDIF
	ENDFUNCTION

	FUNCTION FindNode%:mx%,my%,found AS tFindItem
	LOCAL loop%
	LOCAL cx%,cy%

		self.MouseXYToCellXY(mx%,my%,cx%,cy%)

		// Look for group node
		FOR loop%=0 TO MAX_NODEGROUPS%-1
			FOR loop2%=0 TO BOUNDS(self.nodeGroup[loop%].nodes[],0)-1
				IF self.nodeGroup[loop%].nodes[loop2%].x%=cx% AND
self.nodeGroup[loop%].nodes[loop2%].y%=cy%
					found.groupIndex%=loop%+NODE_NODE1%
					found.nodeIndex%=loop2%
					RETURN TRUE
				ENDIF
			NEXT
		NEXT

		// Now we look for user cells
		FOR loop%=0 TO MAX_USERGROUPS%-1
			FOR loop2%=0 TO BOUNDS(self.userCells[loop%].nodes[],0)-1
				IF self.userCells[loop%].nodes[loop2%].x%=cx% AND
self.userCells[loop%].nodes[loop2%].y%=cy%
					found.groupIndex%=loop%+NODE_USER%
					found.nodeIndex%=loop2%
					RETURN TRUE
				ENDIF
			NEXT
		NEXT

		RETURN FALSE
	ENDFUNCTION

	//! Move a node or user cell
	FUNCTION MoveItem%:mx%,my%,found AS tFindItem
		IF found.groupIndex%>=NODE_NODE1% AND found.groupIndex%<=NODE_NODE10%
			// Moving a node
			self.AddToUndo("Moving Node")
			self.MouseXYToCellXY(mx%,my%,self.nodeGroup[found.groupIndex%-
NODE_NODE1%].nodes[found.nodeIndex%].x%,self.nodeGroup[found.groupIndex%-
NODE_NODE1%].nodes[found.nodeIndex%].y%)
			self.CheckInRange(self.nodeGroup[found.groupIndex%-
NODE_NODE1%].nodes[found.nodeIndex%].x%,self.nodeGroup[found.groupIndex%-
NODE_NODE1%].nodes[found.nodeIndex%].y%)
```

```
		ELSEIF found.groupIndex%>=NODE_USER%
			// User cell
			self.AddToUndo("Movinf User Cell")
			self.MouseXYToCellXY(mx%,my%,self.userCells[found.groupIndex%-
NODE_USER%].nodes[found.nodeIndex%].x%,self.userCells[found.groupIndex%-
NODE_USER%].nodes[found.nodeIndex%].y%)
			self.CheckInRange(self.userCells[found.groupIndex%-
NODE_USER%].nodes[found.nodeIndex%].x%,self.userCells[found.groupIndex%-
NODE_USER%].nodes[found.nodeIndex%].y%)
		ELSE
			DDgui_msg("Invalid group index value : "+found.groupIndex%,FALSE)
		ENDIF
	ENDFUNCTION

	FUNCTION ReturnHandleXYAsString$:
		RETURN self.handleX%+","+self.handleY%
	ENDFUNCTION

	//! Convert all the X/Y values to a string, subtracting the handle X & Y positions from it
	FUNCTION ReturnNodesAsString$:index%
	LOCAL temp$
	LOCAL loop AS tNode

		temp$=""
		FOREACH loop IN self.nodeGroup[index%].nodes[]
			temp$=temp$+(loop.x%-self.handleX%)+","+(loop.y%-self.handleY%)+","
		NEXT

		RETURN LEFT$(temp$,LEN(temp$)-1)
	ENDFUNCTION

	//! Convert all the X/Y values to a string, subtracting the handle X & Y positions from it.
This is for user cells
	FUNCTION ReturnUserCellsAsString$:index%
	LOCAL temp$
	LOCAL loop AS tNode

		temp$=""
		FOREACH loop IN self.userCells[index%].nodes[]
			temp$=temp$+(loop.x%-self.handleX%)+","+(loop.y%-self.handleY%)+","
		NEXT

		RETURN LEFT$(temp$,LEN(temp$)-1)
	ENDFUNCTION

	//! Return a string containing only groups in use
	FUNCTION ReturnGroupsInUse$:
	LOCAL temp$
	LOCAL loop%

		temp$=""
		FOR loop%=NODE_NODE1% TO NODE_NODE10%
			IF BOUNDS(self.nodeGroup[loop%-NODE_NODE1%].nodes[],0)>0
				temp$=temp$+(loop%-NODE_NODE1%)+","
			ENDIF
		NEXT

		RETURN LEFT$(temp$,LEN(temp$)-1)
	ENDFUNCTION

	//! Search for any user cells
	FUNCTION ReturnUserGroupInUse$:
	LOCAL loop%
	LOCAL temp$

		temp$=""
		FOR loop%=0 TO MAX_USERGROUPS%-1
			IF BOUNDS(self.userCells[loop%].nodes[],0)>0
				temp$=temp$+loop%+","
			ENDIF
		NEXT

		RETURN LEFT$(temp$,LEN(temp$)-1)
	ENDFUNCTION

	FUNCTION SetHandleFromText%:text$
	LOCAL val$[]
```

```
        IF text$=""
                self.handleX%=INTEGER(self.GRID_WIDTH%/2)
                self.handleY%=INTEGER(self.GRID_HEIGHT%/2)
        ELSE
                DIM val$[0]
                IF SPLITSTR(text$,val$[],",")>=2
                        self.handleX%=INTEGER(val$[0])
                        self.handleY%=INTEGER(val$[1])
                ELSE
                        DDgui_msg("Invalid handle data - reverting to default",FALSE)
                        self.handleX%=INTEGER(self.GRID_WIDTH%/2)
                        self.handleY%=INTEGER(self.GRID_HEIGHT%/2)
                ENDIF
        ENDIF

        RETURN TRUE
ENDFUNCTION

FUNCTION SetDataNodesFromText%:nodeIndex%,text$,currentColour%,lastColour%
LOCAL val$[]
LOCAL loop%,size%
LOCAL node AS tNode

        DIM val$[0]
        size%=SPLITSTR(text$,val$[],",")
        IF size%>0
                IF MOD(size%,2)>0
                        DDgui_msg("Invalid number of data nodes present",FALSE)
                        RETURN FALSE
                ENDIF

                FOR loop%=0 TO size%-2 STEP 2
                        node.x%=INTEGER(val$[loop])+self.handleX%
                        node.y%=INTEGER(val$[loop+1])+self.handleY%
                        node.currentColour%=currentColour%
                        node.lastColour%=lastColour%
                        DIMPUSH self.nodeGroup[nodeIndex%].nodes[],node
                NEXT
        ELSE
                DDgui_msg("No valid node data",FALSE)
                RETURN FALSE
        ENDIF

        RETURN TRUE
ENDFUNCTION

//! Load the user data
FUNCTION SetUserDataFromText%:nodeIndex%,text$,currentColour%,lastColour%
LOCAL val$[]
LOCAL loop%,size%
LOCAL dataSize%         =       2
LOCAL node AS tNode

        IF text$<>""
                size%=SPLITSTR(text$,val$[],",")
                IF MOD(size%,dataSize%)>0
                        DDgui_msg("Invalid amount of data in user data section",FALSE)
                        RETURN FALSE
                ENDIF

                FOR loop%=0 TO size%-2 STEP 2
                        node.x%=INTEGER(val$[loop])+self.handleX%
                        node.y%=INTEGER(val$[loop+1])+self.handleY%
                        node.currentColour%=currentColour%
                        node.lastColour%=lastColour%
                        DIMPUSH self.userCells[nodeIndex%].nodes[],node
                NEXT
        ENDIF

        RETURN TRUE
ENDFUNCTION

//! Flip vector horizontally or vertically
FUNCTION Flip%:isHorizontal%
LOCAL node AS tNode
LOCAL loop%
```

```
        // Process nodes
        self.AddToUndo("Flipping Nodes & User Cells")
        FOR loop%=0 TO MAX_NODEGROUPS%-1
                FOREACH node IN self.nodeGroup[loop%].nodes[]
                        IF isHorizontal%
                                node.x%=(self.GRID_WIDTH%-1)-node.x%
                        ELSE
                                node.y%=(self.GRID_HEIGHT%-1)-node.y%
                        ENDIF
                NEXT
        NEXT

        // Process user cells
        FOR loop%=0 TO MAX_USERGROUPS%-1
                FOREACH node IN self.userCells[loop%].nodes[]
                        IF isHorizontal%
                                node.x%=(self.GRID_WIDTH%-1)-node.x%
                        ELSE
                                node.y%=(self.GRID_HEIGHT%-1)-node.y%
                        ENDIF
                NEXT
        NEXT
ENDFUNCTION

//! Scroll vertically or horizontally
FUNCTION Scroll%:dirX%,dirY%
LOCAL node AS tNode
LOCAL loop%

        // Process nodes
        self.AddToUndo("Scrolling Nodes and User Cells")
        FOR loop%=0 TO MAX_NODEGROUPS%-1
                FOREACH node IN self.nodeGroup[loop%].nodes[]
                        self.Move(node,dirX%,dirY%)
                NEXT
        NEXT

        // Process user cells
        FOR loop%=0 TO MAX_USERGROUPS%-1
                FOREACH node IN self.userCells[loop%].nodes[]
                        self.Move(node,dirX%,dirY%)
                NEXT
        NEXT
ENDFUNCTION

// Move a node in given direction
FUNCTION Move%:node AS tNode,dirX%,dirY%
        INC node.x%,dirX%
        INC node.y%,dirY%
        self.CheckInRange(node.x%,node.y%)
ENDFUNCTION

FUNCTION Rotate%:angle
LOCAL loop%
LOCAL node AS tNode
LOCAL s,c

        self.AddToUndo("Rotating Nodes and User Cells")

        s=SIN(angle)
        c=COS(angle)

        // First, rotate all nodes in all groups
        FOR loop%=0 TO MAX_NODEGROUPS%-1
                FOREACH node IN self.nodeGroup[loop%].nodes[]
                        self.Rotate2(node,s,c)
                NEXT
        NEXT

        // Now rotate all user cells
        FOR loop%=0 TO MAX_USERGROUPS%-1
                FOREACH node IN self.userCells[loop%].nodes[]
                        self.Rotate2(node,s,c)
                NEXT
        NEXT
ENDFUNCTION
```

```
        //! Perform rotating action
        FUNCTION Rotate2:node AS tNode,s,c
        LOCAL x,y,rx,ry

                x=(node.x%-self.handleX%)*1.0
                y=(node.y%-self.handleY%)*1.0

                rx=(c*x)-(s*y)
                ry=(c*y)+(s*x)
                node.x%=self.handleX%+rx
                node.y%=self.handleY%+ry
                self.CheckInRange(node.x%,node.y%)
        ENDFUNCTION

        //! Clear all data
        FUNCTION ClearData%:
        LOCAL loop%

                FOR loop%=0 TO MAX_NODEGROUPS%-1
                        DIM self.nodeGroup[loop%].nodes[0]
                NEXT

                FOR loop%=0 TO MAX_USERGROUPS%-1
                        DIM self.userCells[loop%].nodes[0]
                NEXT

                // Clear the undo buffer
                DIM self.undo[0]
        ENDFUNCTION

        //! Make sure X & Y values are in range
        FUNCTION CheckInRange%:BYREF x%,BYREF y%
                IF x%<=0
                        x%=0
                ELSEIF x%>=self.GRID_WIDTH%
                        x%=self.GRID_WIDTH%-1
                ENDIF

                IF y%<=0
                        y%=0
                ELSEIF y%>=self.GRID_HEIGHT%
                        y%=self.GRID_HEIGHT%-1
                ENDIF
        ENDFUNCTION

        //! See if there is anything to save.  At least 2 nodes in at least one group
        FUNCTION AnythingToSave%:
        LOCAL loop%

                FOR loop%=0 TO MAX_NODEGROUPS%-1
                        IF BOUNDS(self.nodeGroup[loop%].nodes[],0)>=2 THEN RETURN TRUE
                NEXT

                RETURN FALSE
        ENDFUNCTION

        //! Store in undo array
        FUNCTION AddToUndo%:name$
        LOCAL un AS tUndo

                un.name$=name$
                un.nodeGroup[]=self.nodeGroup[]
                un.userCells[]=self.userCells[]
                DIMPUSH self.undo[],un
        ENDFUNCTION

        //! Undo last action
        FUNCTION DoUndo%:
                IF BOUNDS(self.undo[],0)>0
                        self.nodeGroup[]=self.undo[-1].nodeGroup[]
                        self.userCells[]=self.undo[-1].userCells[]
                        DIMDEL self.undo[],-1
                ENDIF
        ENDFUNCTION
ENDTYPE
```

```
//! Deals with all GUI stuff
TYPE TGui
	WINDOW_WIDTH%
	WINDOW_HEIGHT%
	authorNameWidget$				=	"Author Name :"
	authorNameText$					=	"authorNameText"
	objectNameWidget$				=	"Object Name :"
	objectNameText$					=	"objectNameText"
	actionTypeWidget$				=	"Action Type :"
	actionTypeCombo$				=	"actionType"
	nodeTypeWidget$					=	"Node Type :"
	nodeListCombo$					=	"nodeListCombo"
	nodeList$
	loadData$					=	"Load Data"
	saveData$					=	"Save Data"
	clearData$					=	"Clear Data"
	rotateClockwise$				=	"Rotate Right"
	rotateAntiClockwise$			=	"Rotate Left"
	horizontalFlip$					=	"Horizontal Flip"
	verticalFlip$					=	"Vertical Flip"
	scrollUp$					=	"Scroll Up"
	scrollDown$					=	"Scroll Down"
	scrollLeft$					=	"Scroll Left"
	scrollRight$					=	"Scroll Right"
	undo$						=	"Undo"

	cellInfo[] AS tCellInfo		// Contains cell information

	FUNCTION Initialise%:screenWidth%,screenHeight%
	LOCAL loop%,name$,maxValue%,r%,g%,b%

		DIM self.cellInfo[0]

		self.WINDOW_WIDTH%=screenWidth%/2
		self.WINDOW_HEIGHT%=screenHeight%

		// Setup all cell information.  First, add all the node groups
		FOR loop%=0 TO MAX_NODEGROUPS%-1
			self.AddCellInfo("Node Group #"+loop%,RGB(255-(loop%*15),0,0),RGB(0,255-
(loop%*15),0),NODE_NODE1%+loop%)
		NEXT

		// Now add user cells
		RESTORE parts
		FOR loop%=0 TO MAX_USERGROUPS%-1
			READ name$,maxValue%,r%,g%,b%
			self.AddCellInfo(name$,RGB(r%,g%,b%),0,NODE_USER%+loop%,maxValue%)
		NEXT

		// Build cell list information
		self.nodeList$=self.BuildCellInfo$()
		RETURN TRUE
	ENDFUNCTION

	//! Add cell information data to array
	FUNCTION AddCellInfo%:name$,currentColour%,lastColour%,nodeIndex%,maxRange%=-1
	LOCAL cInfo AS tCellInfo

		cInfo.name$=name$
		cInfo.currentColour%=currentColour%
		cInfo.lastColour%=lastColour%
		cInfo.nodeIndex%=nodeIndex%
		cInfo.maxRange%=maxRange%
		DIMPUSH self.cellInfo[],cInfo
		RETURN TRUE
	ENDFUNCTION

	FUNCTION BuildCellInfo$:
	LOCAL temp$
	LOCAL loop AS tCellInfo

		FOREACH loop IN self.cellInfo[]
			temp$=temp$+loop.name$+"|"
		NEXT

		RETURN LEFT$(temp$,LEN(temp$)-1)
	ENDFUNCTION
	FUNCTION CreateWindow%:
		DDgui_pushdialog(self.WINDOW_WIDTH%,0,self.WINDOW_WIDTH%,self.WINDOW_HEIGHT%)
		DDgui_set("","TEXT","Vector Editor")
		RETURN TRUE
	ENDFUNCTION

	FUNCTION AddOptions%:grid AS TGrid
		DDgui_widget(self.authorNameWidget$,self.authorNameWidget$,self.WINDOW_WIDTH%,0);
			DDgui_text(self.authorNameText$,"",self.WINDOW_WIDTH%-8,0)
		DDgui_widget(self.objectNameWidget$,self.objectNameWidget$,self.WINDOW_WIDTH%,0);
			DDgui_text(self.objectNameText$,"",self.WINDOW_WIDTH%-8,0)

		DDgui_widget(self.actionTypeWidget$,self.actionTypeWidget$,self.WINDOW_WIDTH%,0)
		DDgui_combo(self.actionTypeCombo$,"Add|Move|Remove|Insert",self.WINDOW_WIDTH%-24,0)

		// Set the working mode
		DDgui_set(self.actionTypeCombo$,"SELECT",MODE_ADD%)
		grid.SetActionType(MODE_ADD%)

		DDgui_widget(self.nodeTypeWidget$,self.nodeTypeWidget$,self.WINDOW_WIDTH%,0)
		DDgui_combo(self.nodeListCombo$,self.nodeList$,self.WINDOW_WIDTH%-24,0)
		DDgui_set(self.nodeListCombo$,"SELECT",0)
		grid.SetNodeInfo(self.cellInfo[0])

		DDgui_button(self.loadData$,self.loadData$,(self.WINDOW_WIDTH%/2)-4,0);
				DDgui_button(self.saveData$,self.saveData$,(self.WINDOW_WIDTH%/2)-4,0)
		DDgui_button(self.rotateAntiClockwise$,self.rotateAntiClockwise$,(self.WINDOW_WIDTH%/2)-
4,0);	DDgui_button(self.rotateClockwise$,self.rotateClockwise$,(self.WINDOW_WIDTH%/2)-4,0)
		DDgui_button(self.horizontalFlip$,self.horizontalFlip$,(self.WINDOW_WIDTH%/2)-4,0);
			DDgui_button(self.verticalFlip$,self.verticalFlip$,(self.WINDOW_WIDTH%/2)-4,0)
		DDgui_button(self.scrollUp$,self.scrollUp$,(self.WINDOW_WIDTH%/2)-4,0);
				DDgui_button(self.scrollDown$,self.scrollDown$,(self.WINDOW_WIDTH%/2)-
4,0);
		DDgui_button(self.scrollLeft$,self.scrollLeft$,(self.WINDOW_WIDTH%/2)-4,0);
				DDgui_button(self.scrollRight$,self.scrollRight$,(self.WINDOW_WIDTH%/2)-
4,0);
		DDgui_button(self.clearData$,self.clearData$,self.WINDOW_WIDTH%-6,0)

		DDgui_button(self.undo$,self.undo$,self.WINDOW_WIDTH%-6,0)
		RETURN TRUE
	ENDFUNCTION

	FUNCTION Process%:grid AS TGrid
	LOCAL index%

		IF DDgui_get(self.actionTypeCombo$,"CLICKED")
			index%=DDgui_get(self.actionTypeCombo$,"SELECT")
			IF index%>=0
				grid.SetActionType(index%)
			ELSE
				DDgui_msg("No action type to select",FALSE)
			ENDIF
		ELSEIF DDgui_get(self.nodeListCombo$,"CLICKED")
			index%=DDgui_get(self.nodeListCombo$,"SELECT")
			IF index%>=0
				grid.SetNodeInfo(self.cellInfo[index%])
			ELSE
				DDgui_msg("No group type to select",FALSE)
			ENDIF
		ELSEIF DDgui_get(self.saveData$,"CLICKED")
			self.SaveData(grid)
		ELSEIF DDgui_get(self.loadData$,"CLICKED")
			self.LoadData(grid)
		ELSEIF DDgui_get(self.horizontalFlip$,"CLICKED")
			grid.Flip(TRUE)
		ELSEIF DDgui_get(self.verticalFlip$,"CLICKED")
			grid.Flip(FALSE)
		ELSEIF DDgui_get(self.scrollUp$,"CLICKED")
			grid.Scroll(0,-1)
		ELSEIF DDgui_get(self.scrollDown$,"CLICKED")
			grid.Scroll(0,1)
		ELSEIF DDgui_get(self.scrollLeft$,"CLICKED")
			grid.Scroll(-1,0)
		ELSEIF DDgui_get(self.scrollRight$,"CLICKED")
			grid.Scroll(1,0)
		ELSEIF DDgui_get(self.rotateAntiClockwise$,"CLICKED")
```

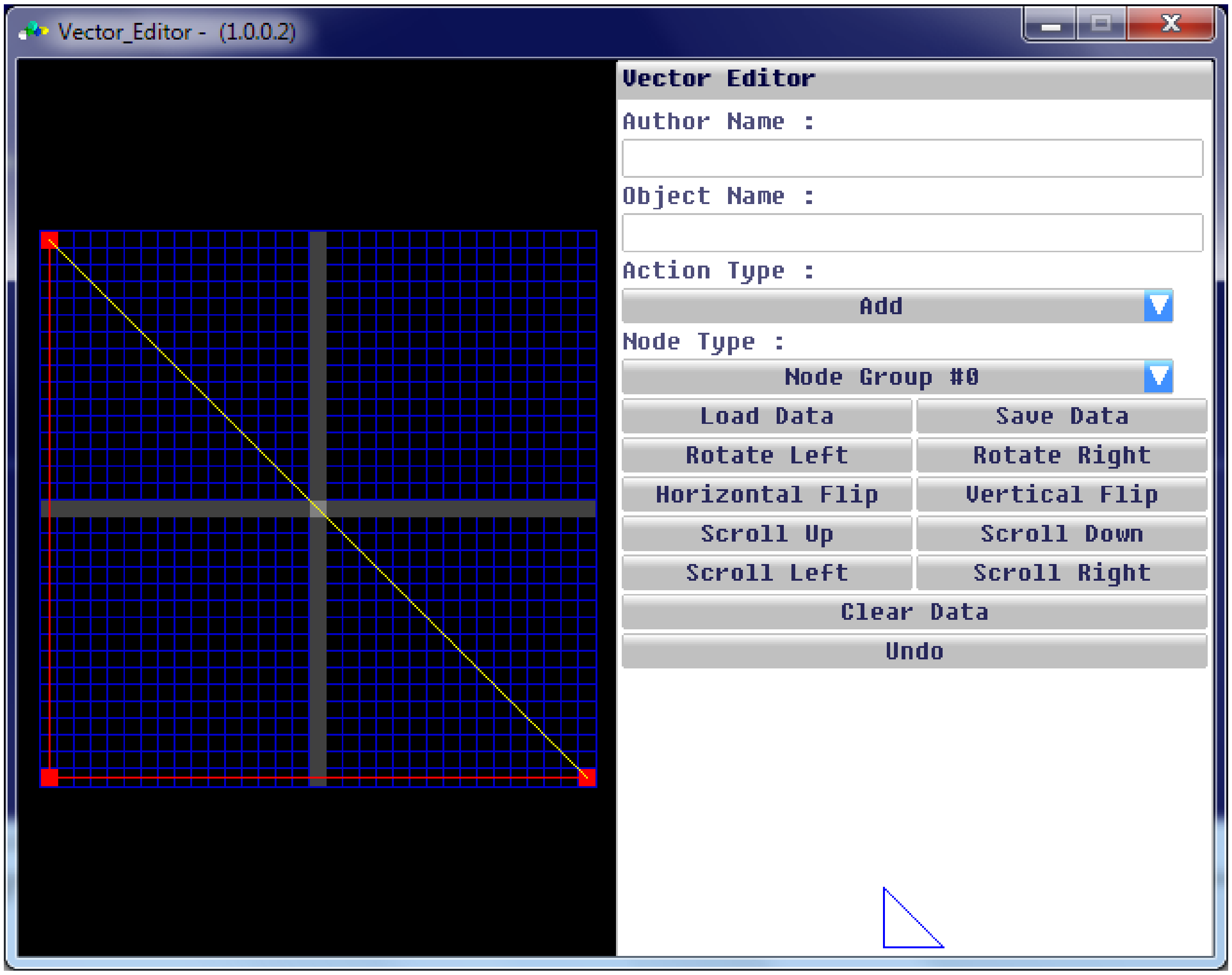
```
                        grid.Rotate(-45.0)
                ELSEIF DDgui_get(self.rotateClockwise$,"CLICKED")
                        grid.Rotate(45.0)
                ELSEIF DDgui_get(self.clearData$,"CLICKED")
                        IF DDgui_msg("Are you want to clear any and all data ?",TRUE)
                                grid.ClearData()
                                DDgui_set(self.authorNameText$,"TEXT","")
                                DDgui_set(self.objectNameText$,"TEXT","")
                        ENDIF
                ELSEIF DDgui_get(self.undo$,"CLICKED")
                        grid.DoUndo()
                ENDIF
        ENDFUNCTION

        //! Save vector data
        FUNCTION SaveData%:grid AS TGrid
        LOCAL fileName$,temp$
        LOCAL loop%
        LOCAL cInfo AS tCellInfo

                // Check to see if there is anything to save
                IF grid.AnythingToSave()=FALSE
                        DDgui_msg("There is nothing to save",FALSE)
                        RETURN FALSE
                ENDIF

                fileName$=DDgui_FileDialog$(FALSE,"*.VEC")
                IF fileName$="" THEN RETURN FALSE

                IF DOESFILEEXIST(fileName$)
                        IF DDgui_msg("The file already exists.  Do you want to overwrite it ?",TRUE,"*
File Present *")=FALSE THEN RETURN FALSE

                        KILLFILE fileName$
                        IF DOESFILEEXIST(fileName$)
                                DDgui_msg("Unable to overwrite the file.  Must be in use by a
program",FALSE,"* File Error *")
                                RETURN FALSE
                        ENDIF
                ENDIF

                INIOPEN fileName$
                INIPUT SECTION_NAME$,KEY_AUTHOR$,DDgui_get$(self.authorNameText$,"TEXT")
                INIPUT SECTION_NAME$,KEY_OBJECTNAME$,DDgui_get$(self.objectNameText$,"TEXT")
                INIPUT SECTION_NAME$,KEY_HANDLE$,grid.ReturnHandleXYAsString$()
                INIPUT SECTION_NAME$,KEY_INUSE$,grid.ReturnGroupsInUse$()

                FOR loop%=NODE_NODE1% TO NODE_NODE10%
                        temp$=grid.ReturnNodesAsString$(loop%-NODE_NODE1%)
                        IF temp$<>"" THEN INIPUT SECTION_NAME$,KEY_DATA$+(loop%-NODE_NODE1%),temp$
                NEXT

                // Find out whether there are any cells for the user-defined cell group
                INIPUT SECTION_USER$,KEY_INUSE$,grid.ReturnUserGroupInUse$()
                FOR loop%=0 TO MAX_USERGROUPS%-1
                        temp$=grid.ReturnUserCellsAsString$(loop)
                        IF temp$<>"" THEN INIPUT SECTION_USER$,KEY_DATA$+loop%,temp$
                NEXT

                INIOPEN ""
                RETURN TRUE
        ENDFUNCTION

        //! Load vector data
        FUNCTION LoadData%:grid AS TGrid
        LOCAL fileName$,text$,list$[]
        LOCAL index%,error$

                fileName$=DDgui_FileDialog$(TRUE,"*.VEC")
                IF fileName$="" THEN RETURN FALSE

                IF DOESFILEEXIST(fileName$)=FALSE
                        DDgui_msg("The given file doesn't exist",FALSE,"* File Not Present *")
                        RETURN FALSE
                ENDIF

                INIOPEN fileName$
```

```
		DDgui_set(self.authorNameText$,"TEXT",INIGET$(SECTION_NAME$,KEY_AUTHOR$,""))
		DDgui_set(self.objectNameText$,"TEXT",INIGET$(SECTION_NAME$,KEY_OBJECTNAME$,""))
		grid.SetHandleFromText(INIGET$(SECTION_NAME$,KEY_HANDLE$,""))

		TRY
			text$=INIGET$(SECTION_NAME$,KEY_INUSE$,"")
			IF text$="" THEN THROW "No group in use list"

			// Clear everything
			grid.ClearData()

			DIM list$[0]
			IF SPLITSTR(text$,list$[],",")>0
				FOR loop%=0 TO BOUNDS(list$[],0)-1
					index%=INTEGER(list$[loop%])
					text$=INIGET$(SECTION_NAME$,KEY_DATA$+index%,"")
					IF text$<>"" THEN
grid.SetDataNodesFromText(index%,text$,self.cellInfo[index%].currentColour%,self.cellInfo[index%].lastC
olour%)
				NEXT
			ELSE
				THROW "Unable to split list of groups"
			ENDIF

			// Get user cells
			text$=INIGET$(SECTION_USER$,KEY_INUSE$,"")
			DIM list$[0]
			IF SPLITSTR(text$,list$[],",")>0
				FOR loop%=0 TO BOUNDS(list$[],0)-1
					index%=INTEGER(list$[loop%])
					text$=INIGET$(SECTION_USER$,KEY_DATA$+index%,"")
					IF text$<>"" THEN
grid.SetUserDataFromText(index%,text$,self.cellInfo[index%+NODE_USER%].currentColour%,self.cellInfo[ind
ex%+NODE_USER%].lastColour%)
				NEXT
			ENDIF

			INIOPEN ""
		CATCH error$
			INIOPEN ""
			DDgui_msg(error$,FALSE)
			RETURN FALSE
		FINALLY


		RETURN TRUE
	ENDFUNCTION
ENDTYPE


// Main program
LOCAL screenWidth%,screenHeight%
LOCAL _gui AS TGui
LOCAL _grid AS TGrid

	ChangeWindowText(" ("+VERSION$+")")

	GETSCREENSIZE screenWidth%,screenHeight%

	IF _gui.Initialise(screenWidth%,screenHeight%)=FALSE THEN endProgram("TGui failed to
initialise")
	IF _grid.Initialise(screenWidth%,screenHeight%)=FALSE THEN endProgram("TGrid failed to
initialise")

	IF _gui.CreateWindow()=FALSE THEN endProgram("Unable to create window")
	IF _gui.AddOptions(_grid)=FALSE THEN endProgram("Unable to add menu options")

	WHILE TRUE
		DDgui_show(FALSE)
		_gui.Process(_grid)
		_grid.DisplayGrid()
		_grid.Process()
		SHOWSCREEN
		HIBERNATE
	WEND
	END
```

```
//! An error occured during initialisation, so end the program
FUNCTION endProgram%:errorMessage$
	DDgui_msg(errorMessage$,FALSE,"* Error *")
	END
ENDFUNCTION

FUNCTION sortRoutine:a AS tNode,b AS tNode
	IF b.x%>a.x% THEN RETURN -1
	IF b.x%<a.x% THEN RETURN 1

	IF b.y%>a.y% THEN RETURN -1
	IF b.y%<a.y% THEN RETURN 1
	RETURN 0
ENDFUNCTION

INLINE
}
#ifndef WIN32
#undef main
#endif

#ifdef WIN32
	// Windows stuff
typedef struct __RECT {
	long left;
	long top;
	long right;
	long bottom;
	} __RECT;

typedef int HDC;

#define SM_CXSCREEN 0
#define SM_CYSCREEN 1
#define HWND_BOTTOM    ((HWND)1)
#define HWND_NOTOPMOST         ((HWND)(-2))
#define HWND_TOP               ((HWND)0)
#define HWND_TOPMOST   ((HWND)(-1))
#define HWND_DESKTOP   (HWND)0
#define HWND_MESSAGE   ((HWND)(-3))

#define SWP_DRAWFRAME          0x0020
#define SWP_FRAMECHANGED       0x0020
#define SWP_HIDEWINDOW                 0x0080
#define SWP_NOACTIVATE                 0x0010
#define SWP_NOCOPYBITS                 0x0100
#define SWP_NOMOVE                     0x0002
#define SWP_NOSIZE                     0x0001
#define SWP_NOREDRAW           0x0008
#define SWP_NOZORDER           0x0004
#define SWP_SHOWWINDOW                 0x0040
#define SWP_NOOWNERZORDER      0x0200
#define SWP_NOREPOSITION       SWP_NOOWNERZORDER
#define SWP_NOSENDCHANGING     0x0400
#define SWP_DEFERERASE                 0x2000
#define SWP_ASYNCWINDOWPOS     0x4000

#define SW_HIDE              0
#define SW_SHOWNORMAL        1
#define SW_SHOWNOACTIVATE    4
#define SW_SHOW              5
#define SW_MINIMIZE          6
#define SW_SHOWNA            8
#define SW_SHOWMAXIMIZED       11
#define SW_MAXIMIZE                    12
#define SW_RESTORE                     13
#define HORZRES        8
#define VERTRES        10

	//DECLARE_C_ALIAS(ShellExecute,"shell32.dll","ShellExecuteA",(HWND,char *,char *,char *,char
*,int),int);
	extern "C" __stdcall int GetSystemMetrics(int);
	extern "C" __stdcall int GetWindowRect(HWND hWnd,struct __RECT *lpRect);
	extern "C" __stdcall int GetClientRect(HWND hWnd,struct __RECT *lpRect);
	extern "C" __stdcall int SetWindowTextA(HWND hWnd,const char *lpString);
	extern "C" __stdcall HWND GetDesktopWindow(void);
	extern "C" __stdcall int SetWindowPos(HWND hWnd,HWND hWndInsertAfter,int X,int Y,int cx,int
cy,int uFlags);
	extern "C" __stdcall int EnumDisplaySettingsA(const char*, unsigned int, void*);
	extern "C" __stdcall HWND GetForegroundWindow(void);
	extern "C" __stdcall int GetLastError(void);
	extern "C" __stdcall int GetSystemMetrics(int nIndex);
	extern "C" __stdcall HDC GetDC(HWND);
	extern "C" __stdcall int GetDeviceCaps(HDC,int);
#else
	#ifndef IPHONE
		// This is for non-windows platforms
		extern "C" __stdcall void SDL_WM_SetCaption(const char *,const char *);
	#endif
#endif

	extern "C" int rename(const char *_old, const char *_new);
namespace __GLBASIC__{
ENDINLINE

FUNCTION ChangeWindowText: version$
INLINE
	DGStr title;

	title=DGStr(__g_AppName);
#ifdef GLB_DEBUG
	title+=DGStr(" (Debug Mode)");
#endif

	title+=DGStr(" - ")+DGStr(version_Str);


#ifdef IPHONE
	// Do nothing on the iPhone/iPod
#elif _WIN32_WCE
#elif GP2X
#elif GP2XWIZ
#elif XBOXLINUX
#elif PANDORA
#elif WIN32
	::SetWindowTextA((HWND) GLBASIC_HWND(),title.c_str());
#else
	::SDL_WM_SetCaption(t_Str.c_str(), title.c_str());
#endif
ENDINLINE
ENDFUNCTION

FUNCTION returnProgramName$:
INLINE
	return DGStr(__g_AppName);
ENDINLINE
ENDFUNCTION

FUNCTION getDesktopSize%:BYREF deskWidth%,BYREF deskHeight%
INLINE
?IFDEF WIN32
	deskWidth=GetSystemMetrics(SM_CXSCREEN);
	deskHeight=GetSystemMetrics(SM_CYSCREEN);
?ELSE
	deskWidth=0;
	deskHeight=0;
?ENDIF
ENDINLINE
ENDFUNCTION

FUNCTION CentreWindow%:
INLINE
#ifdef WIN32
	struct __RECT window;
	unsigned int deskWidth,deskHeight,windowWidth,windowHeight;
	int flags;
	HWND hWnd;

	hWnd=(HWND) GLBASIC_HWND();
	deskWidth=GetSystemMetrics(SM_CXSCREEN);
	deskHeight=GetSystemMetrics(SM_CYSCREEN);

	if (::GetWindowRect(hWnd,&window))
```

```
       {
              windowWidth=window.right-window.left;
              windowHeight=window.bottom-window.top;

              flags=SWP_SHOWWINDOW;
              if (windowWidth>deskWidth || windowHeight>deskHeight)
              {
                     windowWidth=deskWidth;
                     windowHeight=deskHeight;

              }
              else
              {
                     flags|=SWP_NOSIZE;
              }

              ::SetWindowPos(hWnd,HWND_NOTOPMOST,(deskWidth-windowWidth)>>1,(deskHeight-
windowHeight)>>1, windowWidth, windowHeight, flags);
              }
       else
       {
              DEBUG("GetWindowRect (window) failed");
       }
#else
#endif
ENDINLINE
ENDFUNCTION

FUNCTION renameFile%:old$,new$
INLINE
       (rename(old_Str.c_str(),new_Str.c_str())==0 ? TRUE : FALSE);
ENDINLINE
ENDFUNCTION

FUNCTION RGBR%:colour%
INLINE
       return (DGNat) colour & 255;
ENDINLINE
ENDFUNCTION

FUNCTION RGBG%:colour%
INLINE
       return (DGNat) ((colour>>8) & 255);
ENDINLINE
ENDFUNCTION

FUNCTION RGBB%:colour%
INLINE
       return (DGNat) ((colour>>16) & 255);
ENDINLINE
ENDFUNCTION

STARTDATA parts:
       DATA "Main Gun",1,255,0,0
       DATA "Left Gun",3,255,0,0
       DATA "Right Gun",3,255,0,0
       DATA "Left Rear Gun",3,255,0,0
       DATA "Right Rear Gun",3,255,0,0
       DATA "Left Light",3,255,255,0
       DATA "Right Light",3,255,255,0
       DATA "Missile Tube",4,255,255,128
       DATA "Engine",1,255,0,255
       DATA "General",-1,0,255,255
ENDDATA
```

This allows you to create vector-based graphics, and looks like :



To display the results (once saved), you could use the following routine (which ignores special nodes) :

The Line drawing routine was originally written by Gernot Frisch

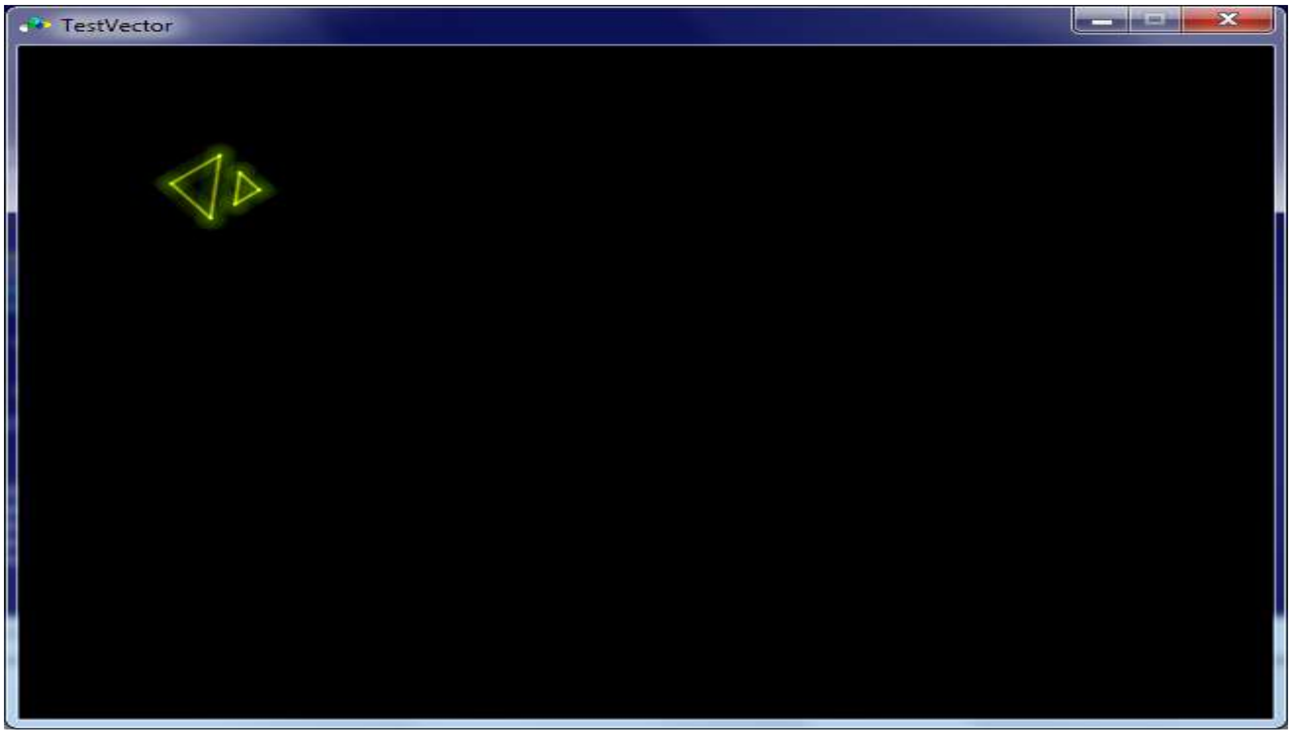
```
// -------------------------------- //
// Project: TestVector
// Start: Friday, July 23, 2010
// IDE Version: 8.036
TYPE tCoord
        x
        y
ENDTYPE

TYPE tGroup
        pos[] AS tCoord
ENDTYPE

TYPE tObject
        author$
        objectName$
        handleX%;handleY%
        groups[] AS tGroup
```

```
ENDTYPE

TYPE TVector
        vectorObject AS tObject

        SECTION_OBJECT$         =       "OBJECT"
        KEY_AUTHOR$             =       "AUTHOR"
        KEY_NAME$               =       "NAME"
        KEY_HANDLE$             =       "HANDLE"
        KEY_INUSE$              =       "INUSE"
        KEY_DATA$               =       "DATA"

        FUNCTION TVector_Initialise%:
                DIM self.vectorObject.groups[0]
        ENDFUNCTION

        FUNCTION TVector_Load%:fileName$
        LOCAL vector AS tCoord
        LOCAL group AS tGroup
        LOCAL temp$,loop%,error$
        LOCAL array$[],dataArray$[]

                IF DOESFILEEXIST(fileName$)
                        TRY
                                INIOPEN fileName$


        self.vectorObject.author$=INIGET$(self.SECTION_OBJECT$,self.KEY_AUTHOR$,"")

        self.vectorObject.objectName$=INIGET$(self.SECTION_OBJECT$,self.KEY_NAME$,"")

                                DIM array$[0]
                                DIM self.vectorObject.groups[0]

                                // Get the handle positions - for information only
                                IF
SPLITSTR(INIGET$(self.SECTION_OBJECT$,self.KEY_HANDLE$,"0,0"),array$[],",")=2
                                        self.vectorObject.handleX%=INTEGER(array$[0])
                                        self.vectorObject.handleY%=INTEGER(array$[1])
                                ELSE
                                        // Ignore anything else and carry on
                                ENDIF

                                // Now we get a list of data sections that are in use
                                DIM array$[0]
                                IF
SPLITSTR(INIGET$(self.SECTION_OBJECT$,self.KEY_INUSE$,""),array$[],",")>0
                                        FOREACH temp$ IN array$[]
                                                DIM dataArray$[0]
                                                DIM group.pos[0]
                                                IF
SPLITSTR(INIGET$(self.SECTION_OBJECT$,self.KEY_DATA$+temp$,""),dataArray$[],",")>0
                                                        FOR loop%=0 TO BOUNDS(dataArray$[],0)-2 STEP 2
                                                                vector.x=INTEGER(dataArray$[loop%])*1.0
                                                                vector.y=INTEGER(dataArray$[loop%+1])*1.0
                                                                DIMPUSH group.pos[],vector
                                                        NEXT

                                                        // Push group into array
                                                        DIMPUSH self.vectorObject.groups[],group
                                                ELSE
                                                        THROW "No data in key : "+self.KEY_DATA$+temp$
                                                ENDIF
                                        NEXT
                                ELSE
                                        THROW "No data to be found!"
                                ENDIF

                                INIOPEN ""
                                RETURN TRUE
                        CATCH error$
                                DEBUG error$+"\n"
                                STDERR error$+"\n"
                                INIOPEN ""
                                RETURN FALSE
                        FINALLY
                ELSE
```

```
                        RETURN FALSE
                ENDIF
        ENDFUNCTION

        FUNCTION TVector_Display%:x,y,angle,scale,colour%,alpha,usePolyVector%=FALSE,useSprite%=-1
        LOCAL group AS tGroup
        LOCAL coords%,size%,loop%
        LOCAL n%,cv,sv

                cv=COS(angle)
                sv=SIN(angle)

                ALPHAMODE alpha
                FOREACH group IN self.vectorObject.groups[]
                        size%=BOUNDS(group.pos[],0)
                        FOR loop%=0 TO size%-1
                                n%=self.TVector_GetNextIndex(loop%,size%)
                                IF usePolyVector%=TRUE
                                        self.Line(x+((cv*group.pos[loop%].x*scale)-
(sv*group.pos[loop%].y*scale)),y+((cv*group.pos[loop%].y*scale)+(sv*group.pos[loop%].x*scale)), _
                                                 x+((cv*group.pos[n%].x*scale)-
(sv*group.pos[n%].y*scale)),y+((cv*group.pos[n%].y*scale)+(sv*group.pos[n%].x*scale)),colour%)
                                ELSE
                                        DRAWLINE x+((cv*group.pos[loop%].x*scale)-
(sv*group.pos[loop%].y*scale)),y+((cv*group.pos[loop%].y*scale)+(sv*group.pos[loop%].x*scale)), _
                                                         x+((cv*group.pos[n%].x*scale)-
(sv*group.pos[n%].y*scale)),y+((cv*group.pos[n%].y*scale)+(sv*group.pos[n%].x*scale)),colour%
                                ENDIF
                        NEXT
                NEXT
        ENDFUNCTION

        FUNCTION TVector_GetNextIndex%:current%,size%
                INC current%
                IF current%>=size% THEN current%=0
                RETURN current
        ENDFUNCTION

        FUNCTION Line: x1, y1, x2, y2, col
        LOCAL c,s,p, w, dx, dy, ddx, ddy, ux, uy, lg

                //line width
                w = 16

                // direction of line
                ddx = x2-x1
                ddy = y2-y1
                lg = SQR(ddx*ddx+ddy*ddy)
                IF lg<0.5 THEN RETURN

                // short caps
                lg=lg*2
                // dir vector
                dx=ddx*w/lg
                dy=ddy*w/lg
                // up vector
                ux=dy
                uy=-dx

                // cap1
                STARTPOLY 0
                        POLYVECTOR x1+ux-dx, y1+uy-dy,  0.5, 0.5,col
                        POLYVECTOR x1-ux-dx, y1-uy-dy,  0.5,63.5,col
                        POLYVECTOR x1-ux,    y1-uy,    31.5,63.5,col
                        POLYVECTOR x1+ux,    y1+uy,    31.5, 0.5,col
                ENDPOLY

                // center
                STARTPOLY 0
                        POLYVECTOR x1+ux, y1+uy, 31.5,  0.5,col
                        POLYVECTOR x1-ux, y1-uy, 31.5, 63.5,col
                        POLYVECTOR x2-ux, y2-uy, 31.5, 63.5,col
                        POLYVECTOR x2+ux, y2+uy, 31.5,  0.5,col
                ENDPOLY

                // cap2
                STARTPOLY 0
```

```
                              POLYVECTOR x2+ux,     y2+uy,     31.5,  0.5,col
                              POLYVECTOR x2-ux,     y2-uy,     31.5, 63.5,col
                              POLYVECTOR x2-ux+dx, y2-uy+dy, 63.5, 63.5,col
                              POLYVECTOR x2+ux+dx, y2+uy+dy, 63.5,  0.5,col
                    ENDPOLY
          ENDFUNCTION
ENDTYPE
```

An example would be :



Part 5

Miscellaneous Routines

These are various routines that you may find useful :

Value Wrapping and limiting range

```
FUNCTION wrapF:Value,minValue,maxValue
LOCAL diff

        IF Value<minValue OR Value>maxValue
                diff=maxValue-minValue
                IF Value<minValue
                        INC Value,diff
                ELSE
                        IF Value>maxValue
                                DEC Value,diff
                        ENDIF
                ENDIF
        ENDIF

        RETURN Value
ENDFUNCTION

FUNCTION wrapI:value%,minValue%,maxValue%
LOCAL diff%

        IF value%<minValue% OR value%>maxValue%
                diff%=maxValue%-minValue%
                IF value%<minValue%
                        INC value%,diff%
                ELSE
                        IF value%>maxValue%
                                DEC value%,diff%
                        ENDIF
                ENDIF
        ENDIF

        RETURN value%
ENDFUNCTION

FUNCTION constrainF:value,minV,maxV
INLINE
        value=(value<=minV ? minV : \
                        value>=maxV ? maxV : value);
ENDINLINE
        RETURN value
ENDFUNCTION
```

Various routines for interfacing with the GLBasic window application. It also includes routines for splitting a RGB value to individual components :

```
INLINE
}
#ifndef WIN32
#undef main
#endif

#ifdef WIN32
        // Windows stuff
typedef struct __RECT {
        long left;
        long top;
        long right;
        long bottom;
        } __RECT;

#define SM_CXSCREEN 0
#define SM_CYSCREEN 1
#define HWND_BOTTOM      ((HWND)1)
#define HWND_NOTOPMOST  ((HWND)(-2))
#define HWND_TOP                ((HWND)0)
#define HWND_TOPMOST    ((HWND)(-1))
#define HWND_DESKTOP    (HWND)0
```

```
#define HWND_MESSAGE    ((HWND)(-3))

#define SWP_DRAWFRAME           0x0020
#define SWP_FRAMECHANGED        0x0020
#define SWP_HIDEWINDOW          0x0080
#define SWP_NOACTIVATE          0x0010
#define SWP_NOCOPYBITS          0x0100
#define SWP_NOMOVE                      0x0002
#define SWP_NOSIZE                      0x0001
#define SWP_NOREDRAW            0x0008
#define SWP_NOZORDER            0x0004
#define SWP_SHOWWINDOW          0x0040
#define SWP_NOOWNERZORDER       0x0200
#define SWP_NOREPOSITION        SWP_NOOWNERZORDER
#define SWP_NOSENDCHANGING      0x0400
#define SWP_DEFERERASE          0x2000
#define SWP_ASYNCWINDOWPOS      0x4000

        extern "C" __stdcall int GetSystemMetrics(int);
        extern "C" __stdcall int GetWindowRect(HWND hWnd,struct __RECT *lpRect);
        extern "C" __stdcall int SetWindowTextA(HWND hWnd,const char *lpString);
        extern "C" __stdcall HWND GetDesktopWindow(void);
        extern "C" __stdcall int SetWindowPos(HWND hWnd,HWND hWndInsertAfter,int X,int Y,int cx,int
cy,int uFlags);
        extern "C" __stdcall int EnumDisplaySettingsA(const char*, unsigned int, void*);
        extern "C" __stdcall HWND GetForegroundWindow(void);
        extern "C" __stdcall int GetLastError(void);
#else
        #ifndef IPHONE
                // This is for non-windows platforms
                extern "C" __stdcall void SDL_WM_SetCaption(const char *,const char *);
        #endif
#endif

extern "C" int rename(const char *_old, const char *_new);
namespace __GLBASIC__{
ENDINLINE

FUNCTION ChangeWindowText: t$
INLINE
#ifdef IPHONE
        // Do nothing on the iPhone/iPod
#elif WIN32
        ::SetWindowTextA((HWND) GLBASIC_HWND(),t_Str.c_str());
#else
        ::SDL_WM_SetCaption(t_Str.c_str(), t_Str.c_str());
#endif
ENDINLINE
ENDFUNCTION

FUNCTION CentreWindow:deskWidth%,deskHeight%
        DEBUG "desktop : "+deskWidth%+" "+deskHeight%+"\n"

INLINE
#ifdef WIN32
        struct __RECT window;
        HWND hWnd;

        hWnd=(HWND) GLBASIC_HWND();
//      DEBUG((int) GLBASIC_HWND());
        if (::GetWindowRect(hWnd,&window))
        {
                ::SetWindowPos(hWnd,HWND_NOTOPMOST,(deskWidth-(window.right-window.left)) / 2,
(deskHeight-(window.bottom-window.top)) / 2, 0, 0, SWP_NOSIZE);
        }
        else
        {
                DEBUG("GetWindowRect failed");
        }
#else
#endif

ENDINLINE
ENDFUNCTION

FUNCTION getCurrentDesktopSize:BYREF width%,BYREF height%
INLINE
```

```
#ifdef WIN32
        struct __RECT desk;
        if (::GetWindowRect(::GetDesktopWindow(),&desk))
        {
                width=desk.right-desk.left;
                height=desk.bottom-desk.top;
                return TRUE;
        }
        else
        {
                return FALSE;
        }
#else
        // Cant currently centre on Linux and Mac
        width=-1;
        height=-1;
        return TRUE;
#endif
ENDINLINE
ENDFUNCTION

FUNCTION renameFile%:dir$,old$,new$
LOCAL ok%
LOCAL tempDir$

        ok%=-1

        INLINE
                tempDir_Str=GETCURRENTDIR_Str();
                if (SETCURRENTDIR(dir_Str.c_str())==TRUE)
                {
                ok=rename(old_Str.c_str(), new_Str.c_str());
        }
        SETCURRENTDIR(tempDir_Str);
        return (ok==0 ? TRUE : FALSE);
    ENDINLINE
ENDFUNCTION

FUNCTION RGBR%:colour%
INLINE
        return (DGNat) colour & 255;
ENDINLINE
ENDFUNCTION

FUNCTION RGBG%:colour%
INLINE
        return (DGNat) ((colour>>8) & 255);
ENDINLINE
ENDFUNCTION

FUNCTION RGBB%:colour%
INLINE
        return (DGNat) ((colour>>16) & 255);
ENDINLINE
ENDFUNCTION

FUNCTION grabScreen%:
LOCAL temp%,sW%,sH%

        temp%=GENSPRITE()
        IF temp%<>-1
                GETSCREENSIZE sW%,sH%
                GRABSPRITE temp%,0,0,sW%-1,sH%-1
        ENDIF

        RETURN temp%
ENDFUNCTION
```

Swap variable values :

```
FUNCTION swapI:BYREF a%,BYREF b%
LOCAL c%

        c%=a%
        a%=b%
        b%=c%
ENDFUNCTION

FUNCTION swapF:BYREF a,BYREF b
LOCAL c

        c=a
        a=b
        b=c
ENDFUNCTION

FUNCTION swapS:BYREF a$,BYREF b$
LOCAL c$

        c$=a$
        a$=b$
        b$=c$
ENDFUNCTION
```

Hexadecimal routine :

```
//! Convert a decimal number to a hexidecimal number
//\param value% - Number of be converted
//\param length% - Number of characters that the result will be padded to
//\return The hexidecimal value as a string
FUNCTION decToHex$:value%,length%=4
LOCAL digit%
LOCAL temp%
LOCAL result$

        IF length%<=0
                RETURN "0"
        ENDIF

        result$=""
        FOR digit%=length% TO 1 STEP -1
                temp%=MOD(value%,16)
                IF temp%<10
                        result$=CHR$(temp%+48)+result$
                ELSE
                        result$=CHR$((temp%-10)+65)+result$
                ENDIF

                value%=value%/16
        NEXT

        RETURN result$
ENDFUNCTION

//! Convert a hexidecimal number of an integer
//\param hex$ - Hexidecimal string
//\return The decimal value of the passed string
FUNCTION hexToDec%:hex$
LOCAL i%
LOCAL j%
LOCAL loop%

        i%=0
        j%=0
        FOR loop%=0 TO LEN(hex$)-1
                 i%=ASC(MID$(hex$,loop%,1))-48
                 IF 9<i%
                        DEC i%,7
                 ENDIF

                 j%=j%*16
                 j%=bOR(j%,bAND(i,15))
        NEXT

        RETURN j
ENDFUNCTION
```

Various maths routines :

```
FUNCTION PI:
       RETURN 3.14159265358979
ENDFUNCTION

FUNCTION PHI:
       RETURN 1.6180339887
ENDFUNCTION

FUNCTION PHI2:
       RETURN 0.6180339
ENDFUNCTION

FUNCTION factorial%:value%
LOCAL total%

       IF value%<=1
              total%=1
       ELSE
              total%=value%
       ENDIF

       WHILE value%>1
              total%=total%*(value%-1)
              DEC value%,1
       WEND

       RETURN total%
ENDFUNCTION

FUNCTION fcmp%:value1,value2,epsilon
       IF ABS(value1-value2)<=ABS(value1)*epsilon
              RETURN TRUE
       ELSE
              RETURN FALSE
       ENDIF
ENDFUNCTION
```